DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

M18327

MacCAD, Computer Aided Design Tool
For System Analysis

by

Kenneth MacDonald

December 1987

Thesis Advisor: George J. Thaler

Approved for public release; distribution is unlimited

T239077

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION NAVAL POSTGRADUATE SCHOOL 6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION NAVAL POSTGRADUATE SCHOOL
6c ADDRESS (City, State, and ZIP Code) MONTEREY, CALIFORNIA 93943-5000	7b ADDRESS (City, State, and ZIP Code) MONTEREY, CALIFORNIA93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

MACCAD COMPUTER AIDED DESIGN TOOL FOR SYSTEM ANALYSIS

12 PERSONAL AUTHOR(S)
MacDonald, Kenneth S.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1987 December	288

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Control Systems, Linear Systems, Computer Aided Design, Graphic Analysis

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

MacCAD, a computer aided design program, was developed as an analysis tool of continuous, linear control systems for the graduate level Controls or Electrical Engineering student. A variety of programs are presently available for the IBM-PC and the IBM Mainframe. MacCAD is written in Pascal and operates on the Apple Macintosh. It was developed in order to meet the need for a powerful yet simple to use program with high quality graphics and multiple window capabilities. It offers a highly flexible block manipulator capable of handling virtually unlimited blocks and loops, along with the standard analysis tools, Bode, Nyquist, Root Locus and Time response plots. Mouse and menu driven, it offers user defined plotting and calculation parameters which include multiple plot overlapping and linear or logarithmic plot point intervals.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL George J. Thaler	22b TELEPHONE (Include Area Code) (408) 646-2056	22c OFFICE SYMBOL Code 62TR

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

MacCAD, Computer Aided Design Tool For System Analysis

by

Kenneth S. MacDonald
Lieutenant, United States Navy
B.S.E.E. United States Naval Academy, 1980

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1987

ABSTRACT

MacCAD, a computer aided design program, was developed as an analysis tool of continuous, linear control systems for the the graduate level Controls or Electrical Engineering student. A variety of programs are presently available for the IBM-PC and the IBM Mainframe. MacCAD is written in Pascal and operates on the Apple Macintosh. It was developed in order to meet the need for a powerful yet simple to use program with high quality graphics and multiple window capabilities. It offers a highly flexible block manipulator capable of handling virtually unlimited blocks and loops, along with the standard analysis tools, Bode, Nyquist, Root Locus and Time response plots. Mouse and menu driven, it offers user defined plotting and calculation parameters which include multiple plot overlapping and linear or logarithmic plot point intervals.

TABLE OF CONTENTS

I. CONTROL SYSTEM ANALYSIS USING MACCAD 8

A. BLOCK DIAGRAMS 8

B. MACCAD CAPABILITIES 9

C. PROGRAMMING PHILOSOPHY 11

II. BASIC MACINTOSH USE 13

A. DESKTOP 13

B. MOUSE 14

C. WINDOWS 15

D. PULL DOWN MENUS 16

E. DESK ACCESSORIES 17

F. KEYBOARD 18

G. DIALOG BOXES 19

H. BASIC PRINTING 22

I. SUMMARY 23

III. STANDARD MACINTOSH MENUS 24

A. BASIC DESCRIPTION 24

B. APPLE MENU 24

C. FILE MENU 25

1. New 25

2. Open 26

3. Save 28

4. Save As 29

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

5. Print 29
6. Quit 31
D. EDIT MENU 31
E PROGRAMMER'S NOTES 32
F. USERS' TIPS 35
IV. BLOCK MANIPULATOR 38
A. BLOCKS AND GROUPS 38
B. PROGRAMMER'S NOTES 41
C. ENTERING DATA 43
D. DISPLAYING BLOCK DATA FOR CHANGING 45
E WEAPON FIRE CONTROL SYSTEM EXAMPLE 47
V. BODE PLOT 67
A. BASIC DESCRIPTION 67
B. USER OPTIONS 68
C. PROGRAMMER'S NOTES. BODE DATA
CALCULATION AND DISPLAY 70
D. BAND PASS FILTER EXAMPLE 74
VI. NYQUIST PLOT 86
A. BASIC DESCRIPTION 86
B. USER OPTIONS 89
C. ILLUSTRATIVE EXAMPLE 1 90
D. ILLUSTRATIVE EXAMPLE 2 99
E PROGRAMMER'S NOTES 104
F. USER'S TIPS 107
VII. ROOT FINDER 111

A. BASIC DESCRIPTION 111
B. A SIMPLE EXAMPLE 111
C. PROGRAMMER'S NOTES: ROOTFINDER PROCEDURE .. 114
VIII. ROOT LOCUS 119
A. BASIC DESCRIPTION 119
B. USER OPTIONS 119
C. FOURTH ORDER
CHARACTERISTIC EQUATION EXAMPLE 121
D. PROGRAMMER'S NOTES 127
IX. TIME RESPONSE 130
A. BASIC DESCRIPTION 130
B. USER OPTIONS 131
C. TIME RESPONSE CALCULATIONS 132
D. ILLUSTRATIVE EXAMPLE 1. THE UNIT STEP INPUT .. 134
E ILLUSTRATIVE EXAMPLE 2. THE RAMP INPUT 137
F. ILLUSTRATIVE EXAMPLE 3. THE SINE WAVE INPUT . 140
G ILLUSTRATIVE EXAMPLE 4. THE IMPULSE INPUT 144
H. PROGRAMMER'S NOTES 148
I. USER'S TIPS 152
X MACCAD SUBROUTINES AND LIBRARIES 155
A. BASIC DESCRIPTION 155
B. SANE LIBRARY 155
C. PROGRAMMER'S EXTENDER 156
D. NUMBERCRUNCH ROUTINES 157
XI. CONCLUSIONS 165
A. SUMMARY OF PROGRAM 165

B. POTENTIAL FOR IMPROVEMENT 166
APPENDIX SOURCE CODE 167
LIST OF REFERENCES 285
INITIAL DISTRIBUTION LIST 286

I. CONTROL SYSTEM ANALYSIS USING MACCAD

A. BLOCK DIAGRAMS.

Most systems studied by the Control Systems student can be considered Single Input Single Output or SISO. As the title suggests, there is only one input and one output. Although there are many measurable states of the system that may be of interest, there is generally only one output that is of prime concern. For example consider a weapon fire control system. Its purpose is simply to aim the weapon at the target. As the target moves the weapon should follow its motion with an acceptable error. Acceptable error means that the weapon projectile will still be effective against the target. The input for this system could be the output from a fire control radar which reports the targets position. The output of the fire control system is the position of the weapon barrel. This would be a SISO system. As mentioned earlier, there may be several states that are of interest. The velocity or acceleration of the barrel during positioning may dictate limitations on the system due to structural strength of the weapon or the power available with which to move it.

Once we have defined the system to analyze, we can use block diagrams to represent it.[1] The block diagram is ideal for this

[1] Block diagrams can of course be used for Multiple Input Multiple Output (MIMO) also.

because it offers symbolic representation in the form of system units or blocks which helps us to understand the various block's functions from their relative positions. Each unit is shown as a block with one input and one output. The relationship between a block's input and output is usually displayed as a LaPlace transfer function shown inside the block. From the properties of LaPlace, we can cascade these unit blocks and calculate the equivalent transfer function by multiplication. Very complex systems can be easily simplified by basic block diagram manipulation techniques. In addition, by displaying each system unit as a separate block, any value in a block's transfer function can be changed and the new system response easily checked.

B. MAC CAD CAPABILITIES.

The program name MacCAD stands for Macintosh Computer Aided Design. It is a program written for the Apple Macintosh[2] personal computer which allows the user to design, analyze and test control systems of his choice. The system data is entered in the form of LaPlace equations for each of the system blocks. The data of the blocks entered can be changed in any way, deleted, or all the blocks can be simplified into a single block to allow for expanding into a larger, more complex system. Regardless of the system complexity, the equivalent transfer function for the entire system is automatically calculated by the program. The system can

[2] Apple Macintosh models supported by MacCAD are the basic Macintosh, Macintosh Enhanced, Macintosh Plus and the Macintosh SE.

then be analyzed by a variety of popular analysis tools. These include Bode Plots, Nyquist Plots, Time response and Root Locus. The Bode Plot shows the output of the system based on a sinusoidal input. It is a cartesian plot with the magnitude and phase delay of the output plotted against the frequency of the input. The Nyquist Plot is a polar plot of the magnitude and phase of the output. Both the Nyquist and Bode frequency plots graphically display a great deal of information such as the systems stability, the speed at which it will respond to an input and as in the case of electrical filters, what frequencies will be passed, which will be attenuated and by how much. MacCAD also provides analysis within the time domain. The response of the system to various types of inputs such as an impulse, step or ramp can also be displayed graphically. The time response shows how quickly the system can respond to these inputs as well as the overshoot, the transient oscillations and the steady state error. Basic transient response characteristics are also obtained from the Root Locus. This is a cartesian plot in the 's' domain of the roots of the characteristic equation of the closed loop transfer function. This plot shows that by changing the gain of the forward path of the system, the roots can be placed in the desired location. By changing various other variables of the system, it can be seen how the response will change. All these tools, and the other utilities available by MacCAD will be discussed in further detail in later chapters.

C. PROGRAMMING PHILOSOPHY.

MacCAD was written in the computer language PASCAL which is the native language of the Apple Macintosh. It was designed and written with the same user friendliness and standard interface philosophy the Macintosh was designed for. The Macintosh gets most of its input from the user through the 'mouse' which is a small cigarette pack sized control that is moved across the table much like a pencil across paper to move the cursor, or pointer on the computer screen. Rather than typing in commands like the IBM, the Mac lets you select the function you want performed with the mouse. Since the commands are not typed in, the commands do not have to be remembered as with other computers. This lets the user concentrate more on what he wants the computer to do rather than how to get the computer to do it. It is this user friendliness that sets MacCAD apart from other system analysis programs.

Prior computer experience is not required to use MacCAD. A few minutes to learn how to use the mouse and the pull-down menus is all that is needed. All tools and utilities are easy to use and self explanatory so the first time user can get desired results without using trial and error. Regardless of what plot you may be looking at or what function you may have just finished using, all of the program's features are still available to you. This means that you will not find yourself in the situation where 'you can't get there from here.' In a word, the Macintosh is not 'modal' which means that you do not have to back track through the menus or commands that got you where you are, to get back to where you

want to be. For example, while viewing the Bode plot, you may want to look at the equivalent transfer function characteristic equation. This evolution is completed on top of the plot displayed. After checking the transfer function, the Bode plot is still there. You may then want to compare your Bode plot findings with the Nyquist plot. You can even display both plots side by side. This is just one of the capabilities that are unique to MacCAD.

II. BASIC MACINTOSH USE

In as much as this document is a technical explanation of the program and it's capabilities, it must also serve as a user's manual. For this reason, the following brief explanation of the Macintosh use is included. It is by no means a substitute for the Apple Macintosh Users Manual but it will contain enough information for the beginner to be able to use MacCAD. It is highly recommended for the user to read the Users Manual as it explains the use of the Mac's clipboard, scrapbook and many other tools which will be of use but will not be discussed here.

A. DESKTOP.

The Mac display is in the form of a desk top. Rather than a textual list of commands and responses. The Mac desktop simulates the working environment. It is initially clear, displaying small graphic images, called Icons with short titles directly under them for each disk presently being used and a trash can in the lower right corner. An icon is an image representation of an application, document[1] or a control to a usable function. They offer quick

[1] Files that represent runable programs are called "applications". Files that are created from a particular application are called "documents." In most cases, any particular document is associated with only one application, the one that created or uses it. With a few exceptions, a document created by one application, cannot be run or used by another application.

recognition as to the type of item they describe and are easier to identify than lists or directories of file names with extensions.

B. MOUSE

The main interface between the user and the Mac is the mouse. The button on the mouse signals to the Mac that it's present location is of specific importance to the user. Moving the mouse moves the cursor or 'pointer' on the screen in the same direction. Positioning the pointer on an item is called 'pointing' to it. The mouse is used to 'select' various items or icons on the screen. Macintosh operation is in the form of 'selection precedes operation.' This means that you identify the item that you want to perform an operation on and then indicate what operation you want performed. When an item is selected, it is highlighted or outlined so you always know what you told the computer to operate on.

An item can be selected in three ways. An icon can be selected by pointing to it and 'clicking,' or pressing and releasing the mouse button. The icon is now selected and highlighted.

Another method of selection is double clicking. This is just like clicking but it is quickly done twice. This does more than just select an item, it also tells the computer to start whatever application or function that item represents. For example, if you double click the mouse on the MacCAD icon, the icon will be selected but the MacCAD application will start running. If a document from an application is double clicked, the application will be run and the document will be loaded into it.

icon of a disk will open a window displaying the contents of the disks in the form of icons[2].

The mouse can also be used to drag across something. This means the button is pressed and held while the mouse is moved. This action is called dragging. When dragging the mouse across the screen, a rectangle is outlined. When the button is released, everything within the rectangle is now highlighted and selected.

Dragging also refers to moving items on the screen. This is done by pointing to an item, pressing and holding the button and moving the mouse. This will also move an outline of the icon selected and when the button is released, the icon is moved to the new location. Depending on where the icon is moved to, various things can happen. For example if a document or application is dragged to the trash in the lower right of the screen, it has been thrown away and is erased from whatever disk it came from. If you drag a something from one disk to another, you have just made a copy of it on the second disk.

C. WINDOWS.

The window mentioned earlier is a rectangular enclosed portion of the screen that lets you 'see' the contents of something. A window will display the contents of the disk being used. It can display applications, documents and folders. Folders are a way of storing and organizing things, much like in a file drawer. Double clicking on a folder will open a window that displays it's contents. Folders can be

[2] Disk and folder contents can be displayed in forms other than icons. Experiment with the View menu on the desktop.

located in disks or within other folders. Usually 8 to 12 windows are the maximum that can be open at any one time. When several windows are displayed at once, they will usually overlap. If the window you want to view is partially covered by another window, pointing anywhere in your window and clicking will select it and make it the active window. This means that it will be displayed on top of all the other windows and you can now do things in that window, such as move or select icons or open other folders. The active window is that which was last opened or used by the mouse. The window that is active can be identified by it's title bar, the bar that goes across the top of the window rectangle. When active, there are several parallel lines across the top. The window can also be moved just like an icon by clicking in the title bar and dragging the window. It can be closed by clicking in the small square in the upper left corner or resized by dragging the square in the lower right corner to the desired size.

D. PULL DOWN MENUS.

Across the top of the screen is the menu bar with the titles of the menus that can be used. Pointing to one of the titles and pressing the button will highlight the title and display the available menu items you can use in the form of a list hanging below the title. The desired menu item is selected by moving the mouse down the list and releasing the button then pointing to the desired item. If a menu item appears dimmed, that means that item is 'inactive' or cannot be used at this time. Selecting it will do nothing. To the right of a menu item you may

see a clover like symbol followed by a single letter. This means there is a keyboard shortcut for selecting that particular menu item. To select that item from the keyboard, you press the 'command' key, the key next to the space bar with the clover symbol on it, and the letter shown in the menu at the same time. The menu item will be selected just as if you did it with the mouse. Typical menu selections that might be made are: OPEN, which will open a folder or disk window or will run an application or document just like double clicking, CLOSE will do the same as clicking in the close box of a window, QUIT will stop the current application from running and will return you to the desktop. In MacCAD you can use menu items to view and manipulate system transfer functions or open windows that contain the various plots discussed in the last chapter. The other menu items will be discussed further in later chapters.

E DESK ACCESSORIES.

There is a special pull down menu located on the left side of the menu bar. The apple symbol is the title for the desk accessories or DAs. The apple menu will always be available, regardless of which application you may be presently running. The items available are like small applications that can run from within the main application. An example of a DA is 'Control Panel' which can be used to change various characteristics of the desktop display, keyboard or mouse operations. Another example is 'MockWrite' which is a small word processor. A variety of calculators are also available. DAs are stored in the system

file[3] rather then the disk containing the application you are running. With DAs you can copy and transport text or graphic displays of nearly anything that appears on the screen for use in other applications. For example, you may want to include a copy of the Bode plot from MacCAD in a lab writeup you are doing with a Macintosh word processor application. This can be done with DAs.[4]

F. KEYBOARD.

Most information in the form of data or text is entered through the keyboard. The Macintosh keyboard is very similar to the basic type writer with the addition of just a few extra keys.[5] The return key, on the right tells the computer that the data just typed in should be accepted now. It may also imply motion after entering the data. For example, after typing in a line of text in a word processor, hitting return causes a carriage return and a line feed. The Enter key, located in the lower right, is similar to the return in that it enters data but usually does not imply any motion. For MacCAD, both keys will function the same by entering data. Motion as mentioned above will not apply to any input for MacCAD. The option keys in the lower left and right corners are not used in MacCAD. The tab key is used to move between data input points for entering data. This will be

---

[3] See Macintosh Users Guide for information on System, Finder and startup disk.

[4] See Macintosh Users Guide for information on Clipboard, Scrapbook, Edit menu items and function keys 3 and 4.

[5] For basic Macintosh. The Mac Plus and SE have several additional keys.

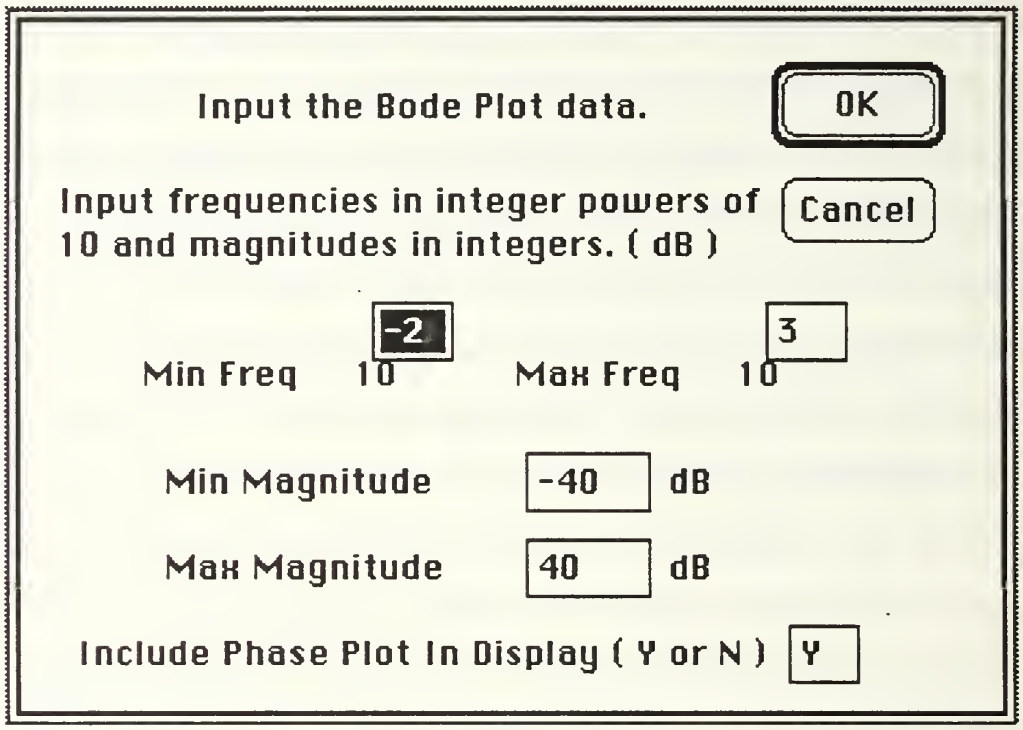
further explained in the next section. This key lets you move from one point to another for adding data or changing data already there. It does not actually enter the data. Only the return or enter keys signal the Macintosh that the data input is complete and should be accepted.

G. DIALOG BOXES.

When the Macintosh requires information from you, it will display a dialog box like the one shown in Fig(1). It will tell you exactly what data is needed and shows you where to enter it. Data insertion points are small boxes in the dialog box that let you type in numbers or text. There are usually several such insertion points in each dialog box. The tab key lets you move from one point to another to enter data. Usually there will already be data in an insertion point box. This is the default data. You can change it if you want but you do not have to. For example, when creating a Bode plot for your system, a dialog box will ask, among other things, what will be the maximum and minimum magnitude values that should be displayed. The values are in decibels and the default values are -40 to 40. Although this range may never have to be changed, there may be conditions where you want to see much higher or much lower values. You can enter what ever values you want and they will become the new default values for any subsequent Bode plots you will make.

Data insertion boxes can also be selected by pointing and clicking with the mouse. Clicking the mouse between two characters in the box will let you insert characters between them. Double clicking a box will highlight the entire number or an entire word if

text is entered.[6] You can also select all or portions of numbers or text by dragging the mouse across the text you want to select. When all or part of text is selected, it will be highlighted. It can be removed by using the delete or backspace key or it can be replaced by typing in whatever you want.


Input the Bode Plot data.
OK
Input frequencies in integer powers of 10 and magnitudes in integers. (dB)
Cancel
Min Freq 10 -2
Max Freq 10 3
Min Magnitude -40 dB
Max Magnitude 40 dB
Include Phase Plot In Display (Y or N) Y

Fig(1) Sample Dialog Box

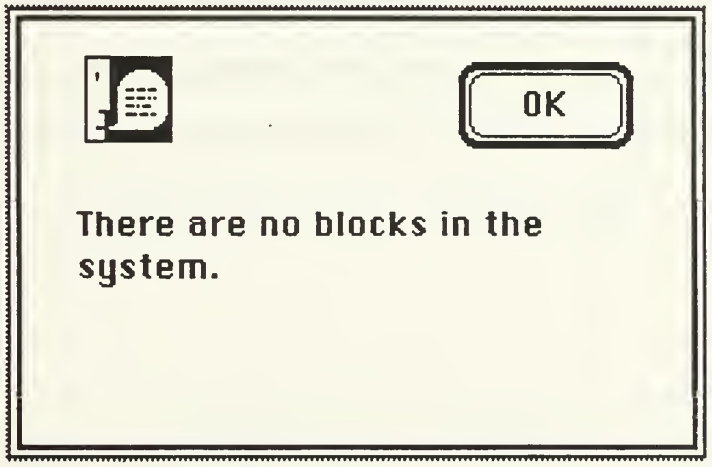
After all data in the insertion boxes is correct, you can enter the data by hitting the return or enter key or by clicking the OK button. In the Bode dialog box, the OK button is the default button because it has a heavy outline. Hitting enter or return is the same as clicking the default button. If you click on Cancel, any changes to the data in

[6] Refer to the Macintosh Users Guide for information on text selection and insertion points.

the insertion boxes will not be saved and the operation which called the dialog box will be canceled. Some dialog boxes offer a variety of buttons for your selection. The one that will most often be used will be the default button and will have the heavy outline. The default button can be selected quickly by hitting return or enter from the keyboard or as any other button, it can be selected with the mouse.

If you type in data that does not apply to it's insertion box, such as typing in letters in the Bode magnitude boxes, the dialog box will not disappear when you hit the OK button. Any insertion boxes that have incorrect data in them will become framed and you will have to correct the data before it will be accepted.

An Alert box is similar to a Dialog box but it does not have any insertion boxes though it may have one or more buttons. Alert boxes are used primarily to notify the user of some condition or to get a simple response like Yes or No from the user. Fig(2) shows an alert box from MacCAD that tells you that a block cannot be deleted from the system because no blocks have been entered yet.



Fig(2) Sample Alert Box

In this case, you are not required to enter any data, just acknowledge that you understand. Alert boxes are used to give the user a second chance to change his mind before doing something to make sure that it is what he really wants to do. For example, after throwing an application away[7] you will see an alert box that asks if you are sure that you want to erase that application from your disk.

H. BASIC PRINTING.

There are a few ways to get a printed output. One way is to select the print menu item from the File menu. You will be presented with dialog boxes so you can select how the data will be printed. You can also do a 'screen dump' which will print the contents of the the active window immediately. This is done by holding the command key and the shift keys down and then typing the number '4'. This is a fast way to get printed output of a plot in MacCAD. If the Caps Lock is down, doing the same thing will cause a printout of the entire screen. You can create a MacPaint[8] document by pressing the command and shift keys and typing the number 3. You can take up to 10 of these 'snapshots' and can then alter them with MacPaint for transferring to a word processor for a lab writeup.

---

[7] Dragging it into the trash on the desktop.

[8] An application by Apple for drawing graphics images such as block diagrams.

I. SUMMARY.

A great deal of information has been quickly provided in this chapter. It is highly recommended that the Apple Macintosh User's Guide be read for further clarification of the material presented here in addition to many other features that were not mentioned.

III. STANDARD MACINTOSH MENUS

A. BASIC DESCRIPTION.

As with nearly all programs written for the Macintosh, MacCAD uses the three standard menus: the apple, File and Edit. This is done for two basic purposes. The first is in order to follow the standard Macintosh programming philosophy, as discussed in the first chapter. This allows the experienced user to easily adapt to a new program since many of the operations are already familiar. The second reason is to allow for easy interaction with various other programs and desk accessories. For example, a desk accessory called 'MockWrite' which allows the user to do basic word processing while within another program also uses the edit menu in MacCAD. In addition, the clipboard and scrapbook use the edit menu. These allow the user to transfer graphics generated by MacCAD into other applications.

B. APPLE MENU.

As mentioned in the first chapter, the apple menu, identified by a small apple in the top left corner, is used primarily for desk accessories. It is also used by most programs as a way of offering program information or on-line help to the user. This is usually the first item of the apple menu.

Some programs, like 'Switcher', which allows the user to run two programs at the same time, also use the apple menu by

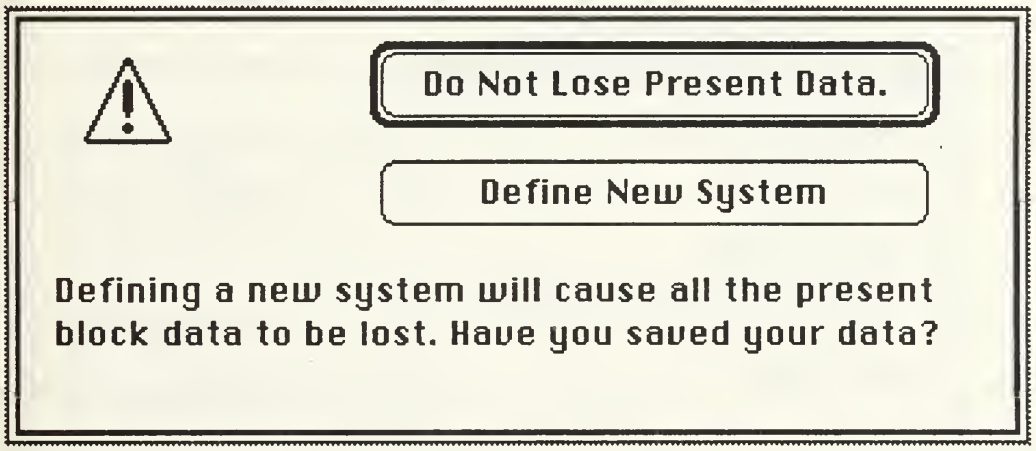
appending additional menu items that let you control their operation. MacCAD works well under 'Switcher' if you set the memory size to at least 256 K by using the 'Configure then install..' item under the 'Switcher' menu. The apple menu is also used by apple's 'MultiFinder' which also allows more than one program to be run at once.

C. FILE MENU.

Most programs offer basically the same items under the 'File' menu. MacCAD offers New, Open, Save, Save As, Print and Quit. The Open, Print and Quit selections also have a keyboard shortcut by holding down the command key, which has a clover leaf symbol on it, at the same time as hitting the first letter of the menu item.

1. New

Selecting New causes the dialog box shown in Fig(1) to be displayed.

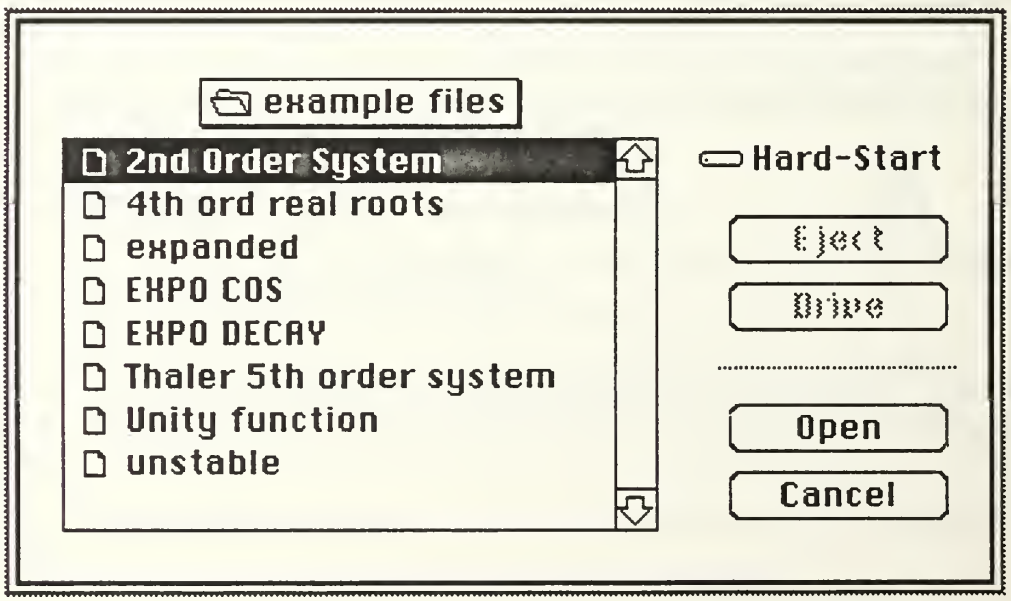

Do Not Lose Present Data.
Define New System
Defining a new system will cause all the present block data to be lost. Have you saved your data?

Fig(1) New Menu Item Dialog Box.

This is a warning to the user that starting a new system will cause the present System block data to be lost. The default selection is the top button which cancels the request. This allows the user to save his present data before starting a new problem. Selecting the lower button 'Define New System' will automatically call the 'Block' menu item 'Add New Block' and you are ready to start inputting the new block data. Any plots that had been drawn are not lost. This allows the user to overlap plots from two different files. The plots cannot be displayed however until at least one block has been entered into the new System block.

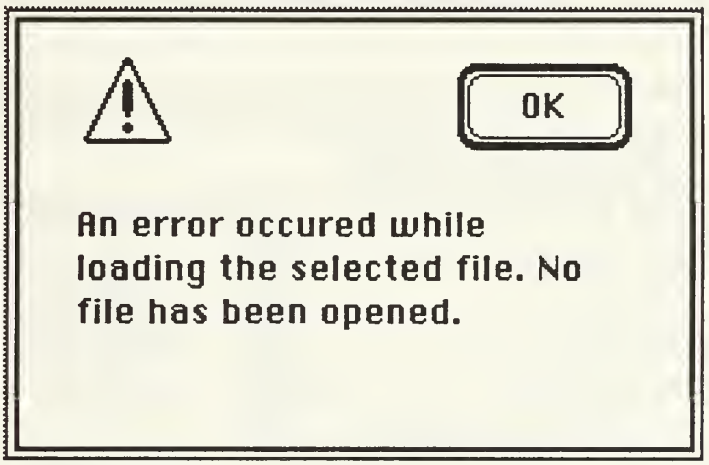
2. Open

The 'Open' item allows the user to load data into the System block that has been previously saved to a file. After selecting 'Open' a dialog box is displayed as in Fig(2).



Fig(2) Open Dialog Box.

This allows the user to select the file to be loaded. Only MacCAD files and folders are displayed in the list box. By clicking on the box that says 'example files' you can move around within the HFS[1] system. Selecting 'Eject' or 'Drive' causes the current disk to be ejected, if a floppy, or the disk drive to be rotated. 'Open' causes the selected folder to be opened or the selected MacCAD document to be loaded. This is also done by double clicking on the selection. Cancel stops the operation and the alert box of Fig(3) is displayed.



Fig(3) Canceling The Open Command.

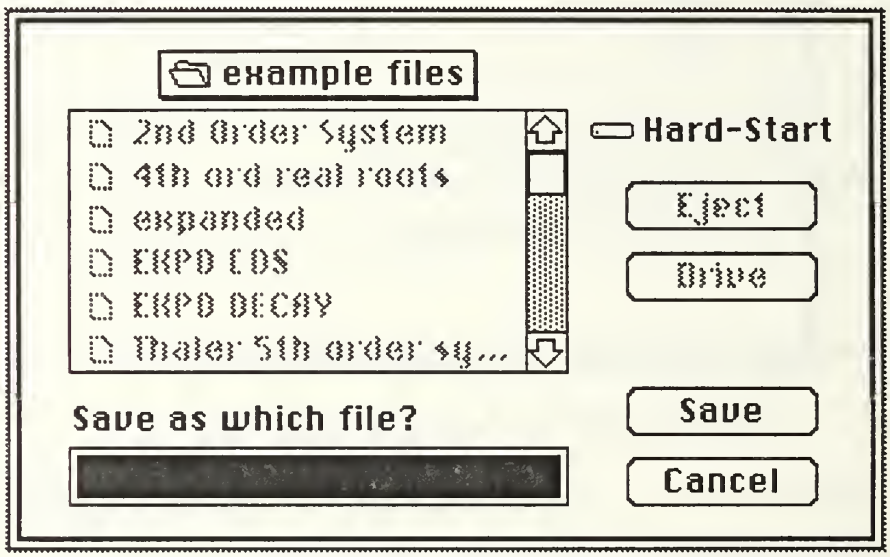
This only tells the user that a file has not been loaded. If a file had been selected by double clicking or clicking the 'Open' button and the alert box of Fig(3) was displayed, it means that there was an error and the selected file was not loaded. This should never happen because the only files that can be selected are MacCAD files but if the selected file had been damaged in some way and was not loaded

[1] HFS - Hierarchical File System. Used on Mac Enhanced, Mac Plus or later models. Displays files and folders only in that folder rather than in the entire disk.

correctly, the user would know. When a new file is opened this way, just as when 'New' is selected, all the previous data, except for the plots, is lost. If the 'Cancel' button is hit, the alert box of Fig(3) will appear and the old data will still be loaded in the system. Before opening a new file be sure any old data has been saved.

3. Save

The 'Save' menu item allows you to save data to a file. The file created is a MacCAD file which can only be used by MacCAD. The dialog box in Fig(4) will be displayed when saving a file.



Fig(4) Saving File Dialog Box.

This dialog box allows the user to select where the new file will be located in the same manner as the file was searched for when opening a file. The new file name is entered in the text box near the bottom. Select a file name that has not yet been used. When 'OK' is selected, if the name has already been used the user will be

asked if the new file being saved should replace the old file that already has that name. If 'Yes' is selected, it will happen. If no is selected, the dialog box will wait for the user to input another file name. If a file has already been saved or was loaded using the 'Open' item and 'Save' is selected, the original file that is located on the disk will be updated to hold the data that is presently in the System block. In this case, no dialog boxes will appear. Saving data is a good practice since a system error could occur and any unsaved data entered would be lost. As new data is entered it should be saved.

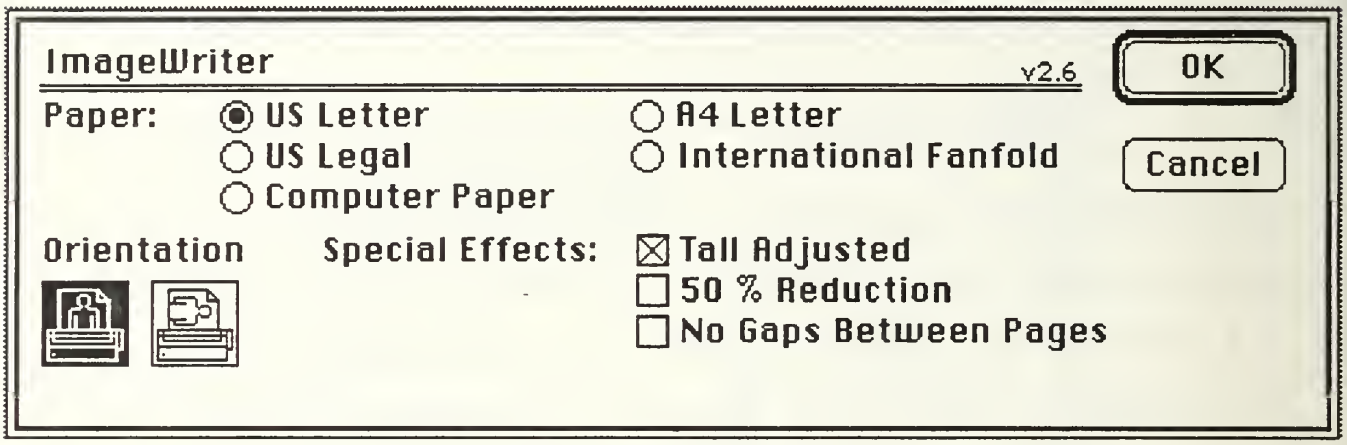
4. Save As

The 'Save As' item is very similar to the 'Save' in that the same dialog boxes are displayed. This item allows the user to save the present version of the System block as a different file. For example, if a file called '2nd Order System' was loaded and changed by adding another pole, it could now be 'Saved As' a file called '3rd Order System.' The original '2nd Order System' is still on the disk and is unchanged but a new file with the latest changes is now saved called '3rd Order System.' This allows several versions to be saved, so the user need not make the same changes over and over again.

5. Print

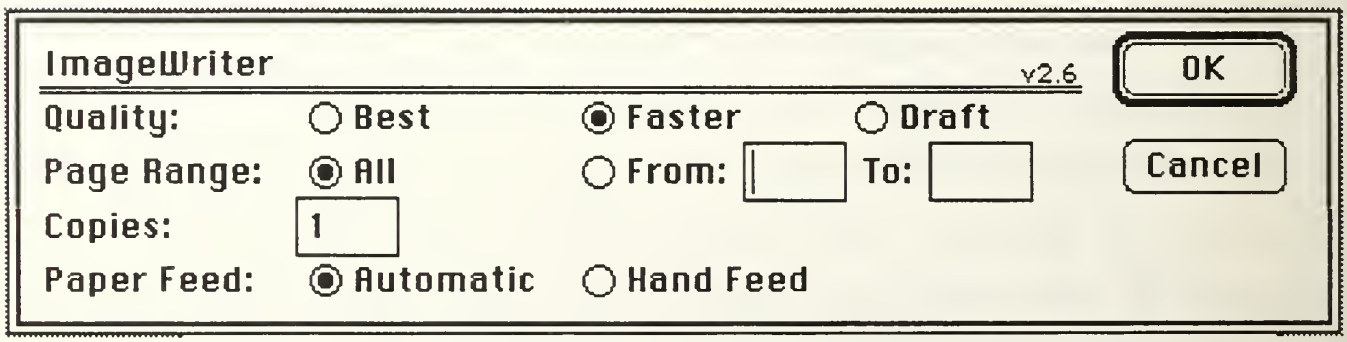
The print command allows the user to get a hard copy of any plot displayed by MacCAD. The plot that is to be printed should be on the front window of the display. After selecting 'Print' the dialog box in Fig(5) is displayed. Under normal conditions, the paper selection should be left at the default setting of 'US Letter'. The Orientation can be set to be vertical (the default) or horizontal which prints the plot

sideways. Under Special Effects, 'Tall Adjusted' should always be selected when printing the Nyquist plot. Otherwise the plot is decreased in width and becomes distorted. Circles on the plot will appear as ovals. For any other plot, the difference is not noticeable. Tall Adjusted is not applicable if the horizontal orientation is selected. 50% Reduction causes the plot to be printed at half it's normal size. This works for both orientations.



Fig(5) Print Dialog Box.

After the proper parameters have been selected and the 'OK' button is selected, the dialog box shown in Fig(6) is displayed.



Fig(6) Second Print Dialog Box.

When printing graphics, 'Faster' is the same as 'Draft'. Selecting 'Best' causes the plot to be double printed. This means the picture is actually printed twice for each carriage return. The result is a darker, higher quality print. The set back is that it takes almost twice as long as the 'Faster' method. The number of copies of the plot can also be set. The default is 1. The Paper Feed default is 'Automatic' which refers to the continuous traction feed paper. 'Hand Feed' can also be selected for printing on other types of paper. Before attempting this method, ensure the printer is properly set up for friction feed.

6. Quit

The last item under the File menu is Quit. Selecting this will cause you to leave MacCAD and return to the desk top. Any data that was not saved when Quit was selected will be lost. You should be sure to save your data before quitting.

D. EDIT MENU.

The edit menu is not directly used by MacCAD. It is included in order to fully use various desk accessories. It also allows the user to transfer text and graphics to and from the clip board. Since these operations are not actually used in MacCAD, no further explanation of the Edit menu will be presented here. Any additional information regarding the edit menu can be found in the Macintosh Users Manual.

E PROGRAMMER'S NOTES.

Files are read from and written to using the functions 'ReadData' and 'WriteData.[2]' They read and write data in the form of segments of data, identified by handles[3]. These procedures work very well with the data structures used in MacCAD since all block and group data is also identified by handles. 'WriteData' reads from an array of handles of the data segments called 'hItem'. Each data segment is sequentially written to the destination file in the same order as they are stored in the array. The order of the array is very important. The association of a block to it's group and a group to it's master block is contained in the order of the handles. This is all handled by the procedure 'SaveSimpBlock'. When saving a file, it is called using sysblockH[4] as the input parameter. This block handle is saved as the first handle in the array 'hItem'. The block's subgroup is then set as the next element in the array. Each of the 5 blocks in that group are checked to see if they are 'used' and if so they are saved as the next array element. This is continued until all 5 blocks are checked. At this time the procedure is done. If the block was 'used', before it is added to the array, it is checked to see if it had been simplified. If so, 'SaveSimpBlock' is called again, from within itself, with the simplified block as it's parameter.

---

[2] Programmer's Extender Vol 2 version 3.05 procedures.

[3] Simply put, a handle is a pointer to a pointer. A pointer is the memory location of an item of interest. A handle is, then, the address of the memory location that holds the address of the memory location of the segment of data of interest.

[4] sysblockH is the handle to the system block.

Each time a simplified block is found, the procedure is called again. It will 'nest' itself as many times as there are layers of simplified blocks within System block.

After all the blocks' handles have been entered into the array, the procedure 'WriteData' is called. This procedure displays the dialog box like that in Fig(4) for the determination of the new file name and location. The same procedures are used for 'Save' and 'Save As'. The only difference is with the SFReply variable, which contains the file name, file type and other information. For 'Save As' and the first 'Save' the SFReply field 'good' is set to false which tells 'WriteData' that a destination file has not yet been determined. In this case, the dialog box is displayed. Otherwise, the file is saved to the same file that it was opened from or last saved to.

Opening a file is done very much like saving a file but in reverse. 'ReadData' is called which displays the dialog box of Fig(2) asking the user to select a file to open. Once selected, the file data is loaded as segments of data identified by the handles in the 'hItem' array. The first handle is always the handle to the System block. It is first converted to a 'bksHdl', a handle to a block. 'sysblockH' is then set to this handle. The procedure 'GetBlockGroup' is then called with 'sysblockH' as it's parameter. It is known that each simplified block in the array is followed by it's subgroup. The next handle in the array is converted to a 'grpHdl', the handle to a group. The blocks 'subgrp', the handle to it's subgroup, is set equal to the second handle in the array. The number of blocks used, in the group is determined

by adding the 'fwdbks' and 'backbks', the number of forward and feedback blocks in the group to get 'numbks', the number of blocks in the group. The next 'numbks' of items in the array are the blocks making up that subgroup. As each is loaded, it is also checked to see if it has been simplified. If so, 'GetBlockGroup' is called from within itself, and the process is repeated. This procedure repeats itself until all the blocks and groups identified by the handles in the array have been loaded.

When 'New' is selected from the 'File' menu, the procedure 'InitBks' from the 'CAD SetUp' module is called. This changes the System group 'fwdbks' and 'backbks' to zero and all the blocks in the group to 'noblock'. 'noblock' is a block that is 'unused', not simplified and has 'no block' for a title.

Selecting 'Print' from the 'File' menu calls the procedure 'DoPrintMenu' from the 'CAD Print Menu' module. This procedure calls 'PrOpen' which initialized the printer manager. A new print record is allocated as 'printH' from the function 'NewPrintHandle'. 'PrStlDialog' and 'PrJobDialog' are each called to display the two print dialog boxes shown in Fig(5) and Fig(6).

If 'OK' is selected in both dialog boxes, the function 'PrintWPic' is called which prints the picture associated with a window, whose handle is input as a parameter. The window that is to be printed is determined from the function 'FrontWindow'. If no window is presently displayed on the screen, neither dialog box will be displayed. Instead, an alert box will instruct the user that a window must be displayed in order to print.

When 'Quit' is selected, the program is terminated simply by falling out of the bottom of the main program loop.

G USERS' TIPS.

When ever opening a new file, check the system block data to ensure numerator and denominator orders are as expected. Particularly when running under the program Switcher, errors can occur when loading file information that goes unnoticed. If the orders are very large or not as expected, try opening the file again. It should work the second time even if it didn't the first.

Saving data as it is being entered cannot be stressed too much. Probably 99% of all experienced computer users have learned this lesson all too painfully after losing hours, maybe days of work because of a system error that wasn't even their fault. Saving the data takes only a second and can be done at any time other than when a dialog box is displayed. The most likely place that a system error would occur is during printing or calculating a new plot. Although no system errors have been noted when running MacCAD by itself, a quick 'Save' is still a good idea.

Save every version of your system. As each significant change is made, use 'Save As' to save that version. Sure enough, if you don't, you will later need to load it again. Even large files created by MacCAD are only about 5 K in size so several files can be made without taking up much room. A 400 K disk with a copy of MacCAD still has enough room for over 50 files. The title of each file is not limited to the 5 or 8 figures that some computer operating systems

require. The Macintosh will accept up to 31 letters or numbers for a title. This means several similar versions can be saved while still being distinguishable.

Each time 'New' or 'Open' is selected from the 'File' menu, additional block data is added to memory. The old data is not erased. This could only be a problem if 'New' or 'Open' has been selected many many times (greater than about 20 for round numbers) and if the Macintosh you are using has less than 1 meg of RAM or you are operating under Switcher or MultiFinder. In all the testing to date, this has not happened but to ensure it's prevention, after selecting 'New' or 'Open' about 20 times, it would be good to find a convenient time to quit from MacCAD and run it again. This erases all the memory containing the old data that is not in use any longer, preventing you from ever running out of memory.

When printing the Nyquist plot, be sure to select 'Tall Adjusted' from the first print dialog box shown in Fig(5). This will ensure the circles will all be round. For all other plots, it may be convenient to not select it as it will leave a larger margin for additional notes after printing.

A fast though not high quality print out of a plot can be obtained by a 'screen dump'. There are two ways this can be done. With the caps lock key locked down, press the shift and command key while pressing the number 4 key. This will cause the printer to print everything that is displayed on the screen. It only takes a few seconds. If the caps lock is not locked down and the same procedure

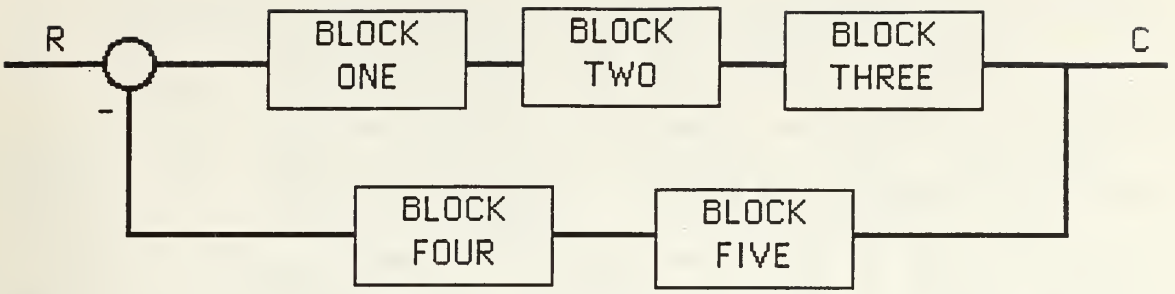
is done, the active window will be printed. This means if two plot windows are displayed on the screen, only the front or active window will be printed. Unlike using the 'Print' menu item, the window screen dump will not print the entire contents of the window, only the part that is presently displayed. Again, this only takes a few seconds and no dialog boxes are needed.

IV. BLOCK MANIPULATOR

The capability of block manipulation is the heart of MacCAD. It allows great flexibility in the entering, editing and simplification of the transfer functions that describe the system.

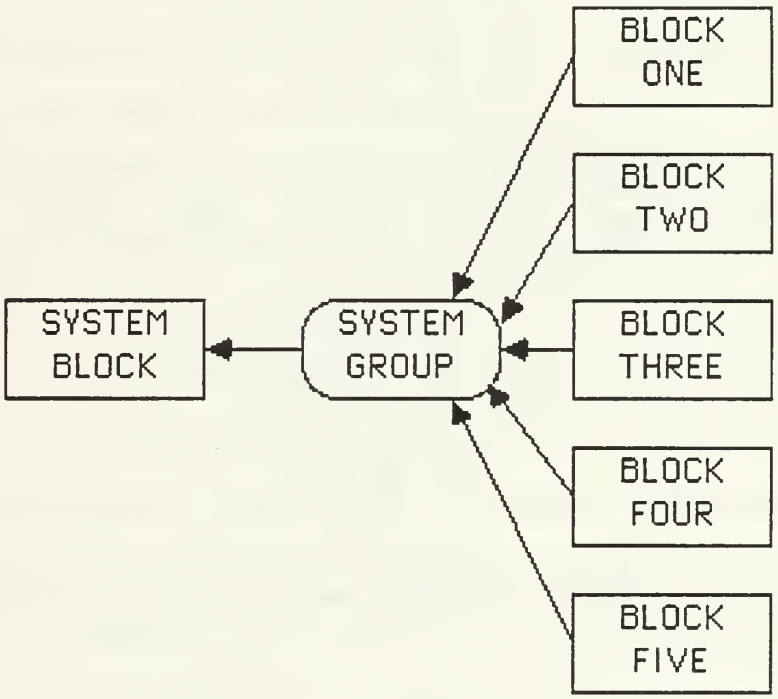
A. BLOCKS AND GROUPS.

Data is stored in the form of blocks and groups. A block is as it appears in a block diagram, a single element in the entire system. Just as a Naval Task Force might be made up of a number of ships, a group is made up of a number of blocks. A block contains a transfer function in the 's' domain made up of a numerator and a denominator with a maximum order of 10. Up to 5 blocks, in a loop path, make up one group. The loop path can be an open loop, closed loop, forward path or Geq. These will be discussed later in this chapter. The number of blocks that form the group will also have an equivalent transfer function, calculated in accordance with the loop path. The equivalent transfer function of this group of blocks is stored in another block. This block is called the System block. It contains the equivalent transfer function of the whole system. The group that the system block represents is the System group. Fig(1) shows a typical block diagram of a system to analyze.



Fig(1) Basic Block Diagram.

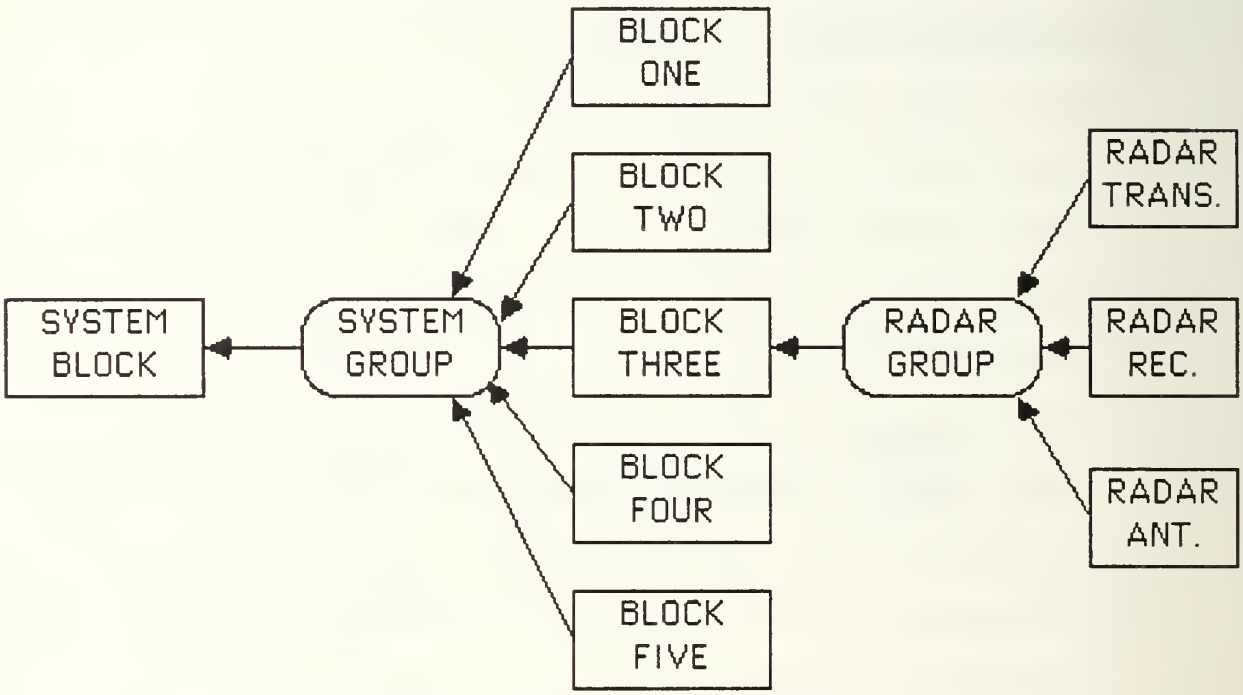
These blocks are again shown in Fig(2) as related to the System block and System group.



Fig(2) Relationship Between Blocks, System Group and System Block.

Just as the system block is the simplification of the five blocks in the system, any one of those five blocks might be the

simplification of yet another group of up to 5 blocks. Using the example started in chapter one, the weapon fire control system, we may want to include the radar in the description of the system. The transmitter, receiver and antenna would all have their own transfer functions to add to the entire system. The radar transfer functions could be simplified into a group which is represented by block three. Fig(3) shows how this 'tree' of blocks and groups can work.



Fig(3) Example of Simplification within the System Group.

Any of the blocks in the original group or in the new radar group could also be simplifications of other groups of blocks. The combinations are endless.

B. PROGRAMMER'S NOTE: BLOCK AND GROUP DATA TYPES.

In order to allow maximum flexibility in the number of blocks in groups, their editing, additions and deletions, related blocks and groups are connected by handles (pointers to pointers). For example, block three in Fig(3) will have a handle to System group because it is a member, as well as a handle to radar group because it is the simplification of the radar group. Likewise, each group has handles to the blocks that it contains as well as a handle to the block that is it's simplification. This may sound redundant at first but it allows you to move from any block to any related group and vice versa. Rather than allocate memory for a set number of groups or blocks this method allows greater flexibility.

Each block contains a transfer function stored as a numerator and denominator in a coefficient form. MacCAD stores polynomials as two possible variable record types. PolyCoef is the variable used to store a polynomial in it's coefficient form. PolyFact stores the polynomial in it's factored form. Both PolyCoef and PolyFact have a field 'degree' for the polynomial order, 'gain' for the multiplier of the whole polynomial and an array of coefficients or 'complex' numbers to describe the equation. The variable type 'complex' is a record of the fields 'realpart', for the real part of a complex pair and 'imagpart' for the imaginary part. There is also a boolean expression 'justreal' that, if true, means the 'complex' variable represents only one real number. If 'justreal' is false, the 'complex' variable represents a pair of complex conjugates in the form of 'realpart'+/-J*'imagpart'.

As mentioned earlier, blocks store their information in PolyCoef form but the data can also be entered or displayed in PolyFact form. This requires the use of a subroutine called RootFinder which solves for the roots of the coefficient polynomial and stores them in a factored polynomial. The block data structure is shown below:

```
block = record
    title:string[16];
    used:boolean;
    changed:boolean;
    num:polycoef;
    den:polycoef;
    factored:boolean;
    forward:boolean;
    simplified:boolean;
    simpform:1..4;
    subgrp:grpHdl;
    fromgrpHdl:grpHdl
end;
```

'title' is a string of text input by the user to identify the block. 'used' is true if this block contains system data and false if the block has been deleted or has not yet been used. 'changed' is a flag used for recalculating group equivalent transfer functions if anything in the block has been changed. 'num' and 'den' are the transfer function numerator and denominator respectively. 'factored' is a flag used to show if the polynomials were originally entered in factored form. 'forward' is true if the block is in the forward path of the group loop, and false if it is in the feedback path. 'simplified' is true if the block is the simplified result of a group of other blocks. If it is simplified, 'simpform' shows the type of loop describing the

group. 'subgrp' is the handle to the group that the block is simplified from. 'fromgrpHdl' is a handle to the group that this block is a member of. The group data structure is:

```
group = record
    ownHdl:grpHdl;
    maingrp:boolean;
    masterblock:bksHdl;
    fwdbks:integer;
    backbks:integer;
    bksused:array[1..5] of bksHdl;
    posFback:boolean;
end;
```

'ownHdl' is the handle to that group. It is used when adding blocks so they know where their 'group' is located. 'maingrp' is true if the group is the system group. 'masterblock' is the handle to the block that is it's simplification. 'fwdbks' and 'backbks' are the number of blocks in the forward and feedback paths. 'bksused' is an array of pointers to the blocks in the group. 'posFback' is true if the system has positive feedback and false if it has negative feedback.

C. ENTERING DATA.

The tools of the block manipulator are in the Blocks pull down menu. They include Change, Add New Block, Simplify and Delete Block. Data in the form of transfer functions can be entered either in coefficient form or in factored form by selecting Add A Block from the Blocks menu. Presently the program can display and input polynomials up to 10th order although the data structures and subroutines have been written to handle 19th order. This limitation is

due to screen size when entering large polynomials. Expansion to completely handle 19th order equations would not be difficult if it was determined necessary.

1. Loop Path

When adding the first block, the loop path for the group is also set. The block is identified as a forward block (in the forward path) or a back block (in the feedback path). Loop paths are defined as one of four possibilities: Forward Path, Open Loop, Closed Loop and Geq. The loop paths are defined in terms of 'G', the product of the forward blocks and 'H', the product of the back blocks. The default loop path is Geq.

Forward Path is defined as 'G'. No feedback of any kind is used. Any back blocks in the system have no effect for this loop path.

Open Loop is the 'G H' product which is the product of all the blocks in the group. It should be noted that the output, therefore, is not taken at the output of the forward block in the farthest right position of the loop as is normally the case. It is taken at the output of the back block in the farthest left position in the loop. The loop is not connected and no feedback actually takes place. This basically lets you observe the signal that would be fed back to the summing junction for the determination of the error signal.

Geq is in the form of Geq = G/(1 + GH) with G and H defined as above. If there are no back blocks in the system, H defaults to zero. In this loop path, the loop is closed only if there is at least one block in the feedback path. If there are no feedback blocks, Geq is the same as Forward Path. Geq is the default loop path.

The Closed Loop path is equal to Geq/(1 + Geq) with Geq defined as above. This is the same as the Geq loop path with unity feedback added. It allows a system with feedback compensation to also have unity feedback. If the system has no back blocks, H still defaults to unity so the total system will then have two unity feedback paths, or an effective feedback of H = 2.0.

D. DISPLAYING BLOCK DATA FOR CHANGING.

After data has been entered, it may be examined at any time. The default display for polynomial data is the coefficient form but any block can be viewed in either form. It should be noted that displaying a polynomial in the factored format requires the roots of the coefficient polynomial to be solved. If it is desired only to observe the roots and not change them, the Cancel button of the dialog box should be clicked. This prevents roundoff error from entering into the original polynomial. If the OK button is clicked, the root values are multiplied to get back to the coefficient form. It can be seen that there are two chances for roundoff error if the OK button is selected. For this reason, the OK button should only be clicked if the roots have been changed.

Selecting 'Change' from the Blocks menu allows you to change any part of any block presently in the system group, either directly or indirectly as a part of a simplified group. In order to select which block to change, you are given the choice as to view the system as a group, which allows you to select which block to look at, or you can see the entire system transfer function as one block. With the former,

you can make changes to blocks presently in the system or add more. With the later, you can only view the system equivalent transfer function, no changes can be made. When viewing the blocks of a group you also have the option to make other changes. You can change the type of feedback, the loop path or you can add another block to the group. When viewing the system as a group of blocks you can view the data in any block and if one of the blocks was simplified from another group of blocks, you can again look at that group as either one simplified block or as the group of blocks that made it. If you look at it as one simplified block, you can make changes to that equivalent block transfer function even though the blocks of the group that made that block were not changed. The system transfer function in System Block will then reflect the changes in the simplified block. The block can be restored to it's original value by viewing one of that block's group of blocks and selecting the OK button, even though you made no changes. This tells MacCAD to recalculate all groups and blocks that require updating.

Blocks are deleted from the system by selecting Delete from the Blocks menu. A block is selected in the same was as when changing a block. Deleting any block will change the equivalent transfer function of the group that it was from. You can still choose to view a group as one simplified block rather than as all of the blocks that made up the group. If you choose this block to delete, all of the blocks that were simplified will also be deleted. This cannot be done with System Block however. After selecting the block to delete, you are given an alert box asking if you are sure that you want to delete

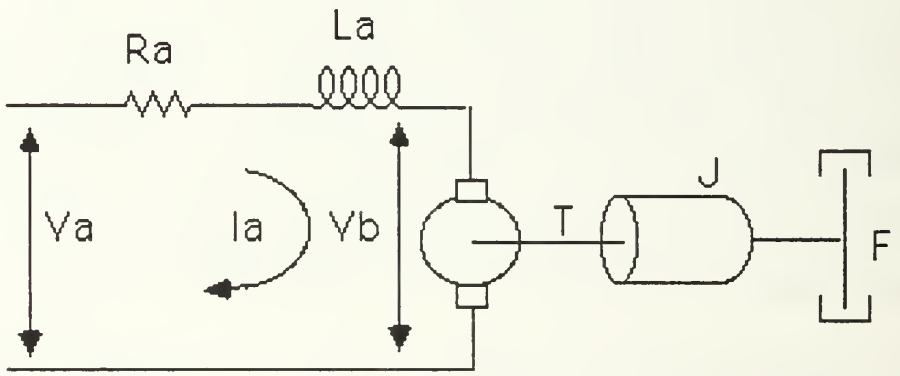
that block. The default response is No. If this is selected, no changes will be made. If Yes is clicked, then the block will be permanently deleted from the system. When asked for which block to delete, you can also make any of the changes available when viewing blocks such as change the feedback type or the loop path.

All the blocks that are presently in System Group can be simplified into one block by selecting Simplify from the Blocks menu. You can enter a title for the new simplified block. More blocks can then be added to System Group, around the newly simplified block. After more blocks are added, they can be simplified again. The only limitation to the number of simplifications is that the order of the numerator and denominator of System Block as mentioned earlier. MacCAD will still calculate the transfer function in the System Block if the order is higher than 10. It just cannot display the data in a simplified block form. Plotting analysis may still be done on systems with orders between 10 and 19 but this should be done with extreme caution. Since MacCAD has not been updated to completely handle the higher order equations, a system crash may occur.

E. WEAPON FIRE CONTROL SYSTEM EXAMPLE.

Use of the block manipulator will be demonstrated by considering the example in chapter one. This is a weapon fire control system that attempts to move the barrel of the weapon so as to follow the moving target. We will assume the weapon is mounted on a rotating turret. We will consider only the rotational angle of the

turret and not it's elevation. The target bearing will be the input to our system in the form of an electric signal with an amplitude proportionate to the angle Θ of the target position. This will be our reference R. The angular position of the turret will be our controlled output C. An armature-controlled dc motor will be used to position the turret. An amplifier will compare the input R to the output C to determine an error. This error will be amplified and used as the input to the dc motor. The actual derivation of the motor equations will be omitted here as they are not the point of this example. The final equations will be used to set up our block diagram. The schematic diagram for the dc motor is shown in Fig(4).



Fig(4) Armature-Controlled DC Motor.

The torque of the motor is proportional to the armature current so

$$T = KIa.$$

The current is calculated by;

$$Ia = (Va - Vb)/(Ra + La\ s)$$

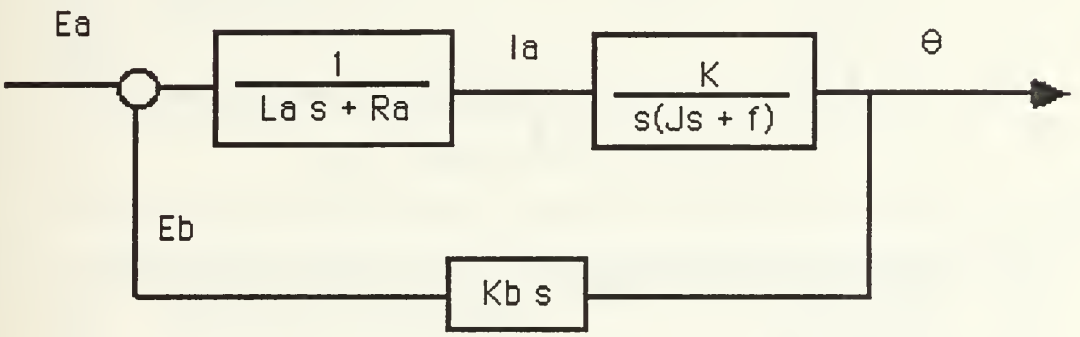
where Vb is the back emf from the motor and

$$Vb = Kb\ d\theta/dt \quad \text{or} \quad Vb(s) = Kb\ s\theta(s)$$

The torque is applied to the inertia and friction so

$$T = J\ d^2\theta/dt^2 + f\ d\theta/dt \quad \text{or} \quad T(s) = Js^2 + fs$$

Using these equations, the block diagram for the motor is shown in Fig(5).



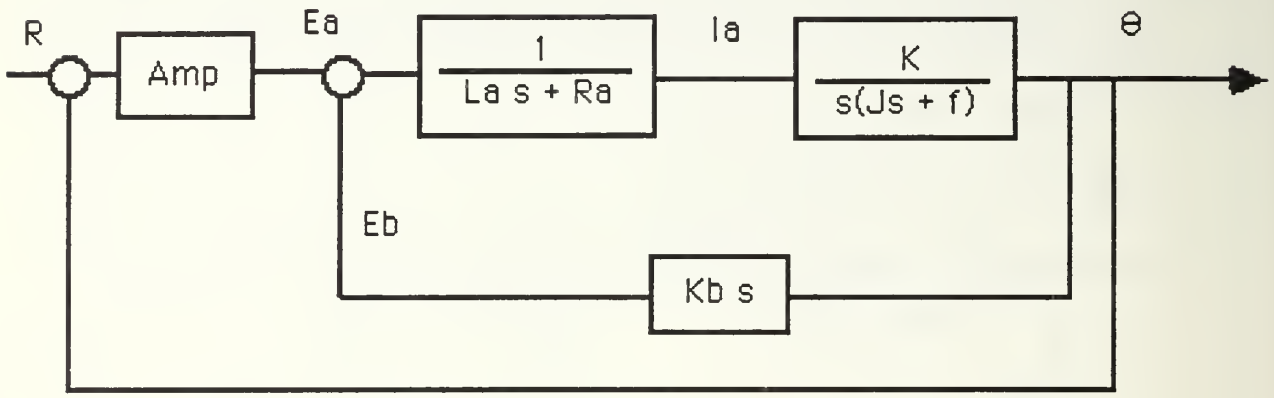
Fig(5) Block Diagram for DC Motor.

The input voltage Ea is actually the error between the turret position and the actual direction of the target. Adding the feedback and the amplification gives the final block diagram in Fig(6).

We will now use the MacCAD Blocks menu items to input and check the fire control system. For the purpose of illustration, the following values will be used.

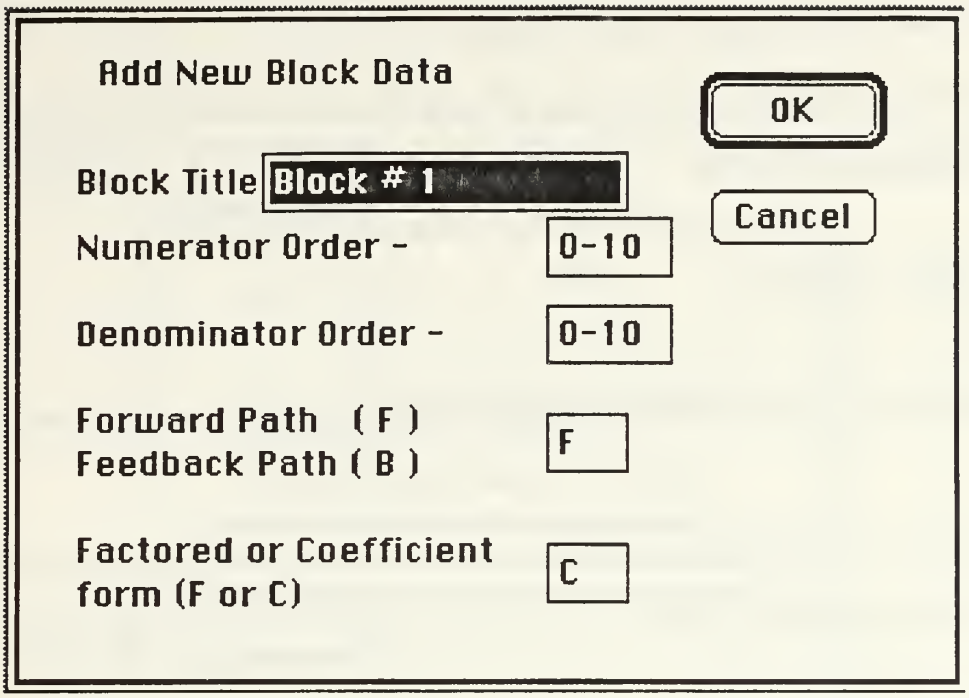
Ra = 100, La = .01, J = 250, f = 10, K = 100, Kb = 2, amp gain = 12.

With this particular block diagram, the blocks can be entered directly into the system. A later example will show how initial changes would have to be made. The transfer functions are input starting from the innermost loop. After starting MacCAD, the Add A Block item is selected from the Blocks menu. Fig(7) shows the dialog box presented.

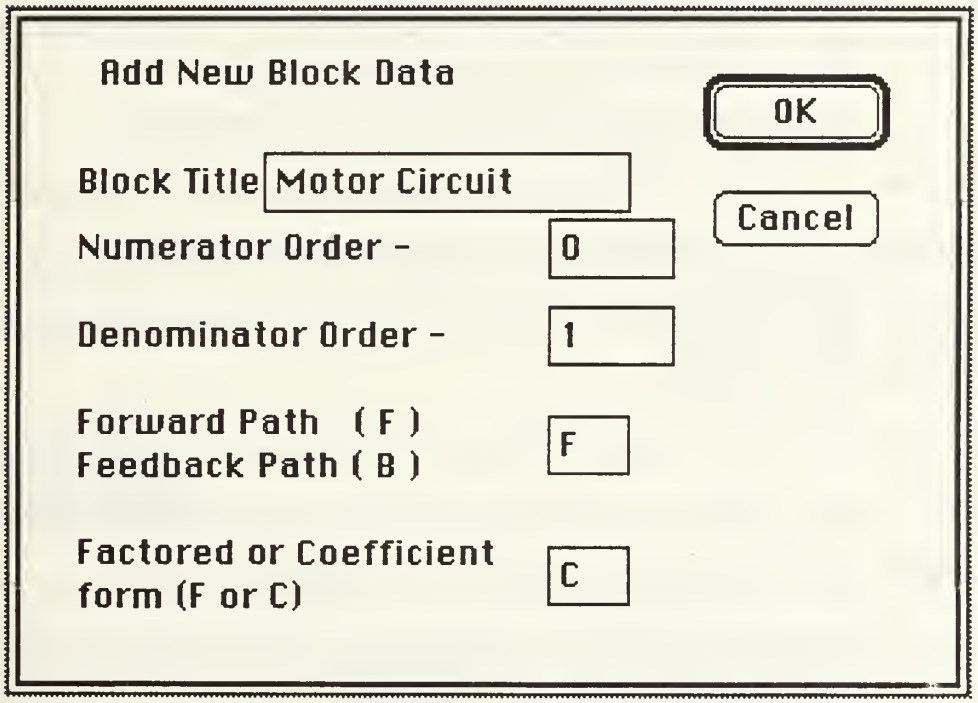


Fig(6) Final Block Diagram for Fire Control System.

Note the default settings of forward and coefficient form. As blocks are added, the default title starts at 'Block One' and increases. It is advisable to change these titles because if two blocks are entered and the first is deleted, the next time Add A Block is called, the default title will be 'Block Two' again and both blocks in the group will have the same title. A title and information describing the top left block in the inner loop in Fig(6) is entered. Before clicking OK, the dialog box looks like Fig(8).

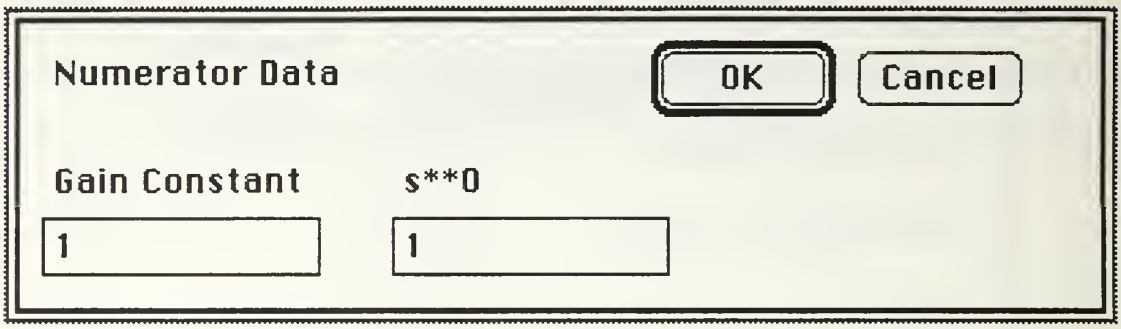


Fig(7) Add A Block Dialog Box.

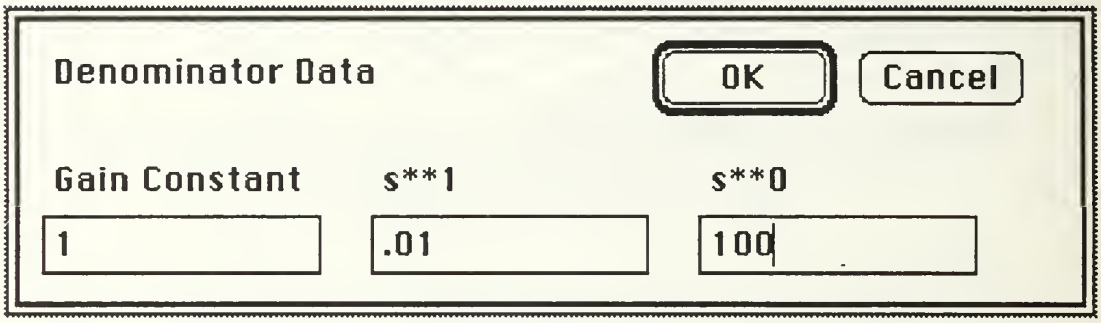


Fig(8) Entering The First Block Data.

After clicking OK the numerator data is entered as in Fig(9) and the denominator data is also entered in Fig(10).



Fig(9) Numerator Data Entered.

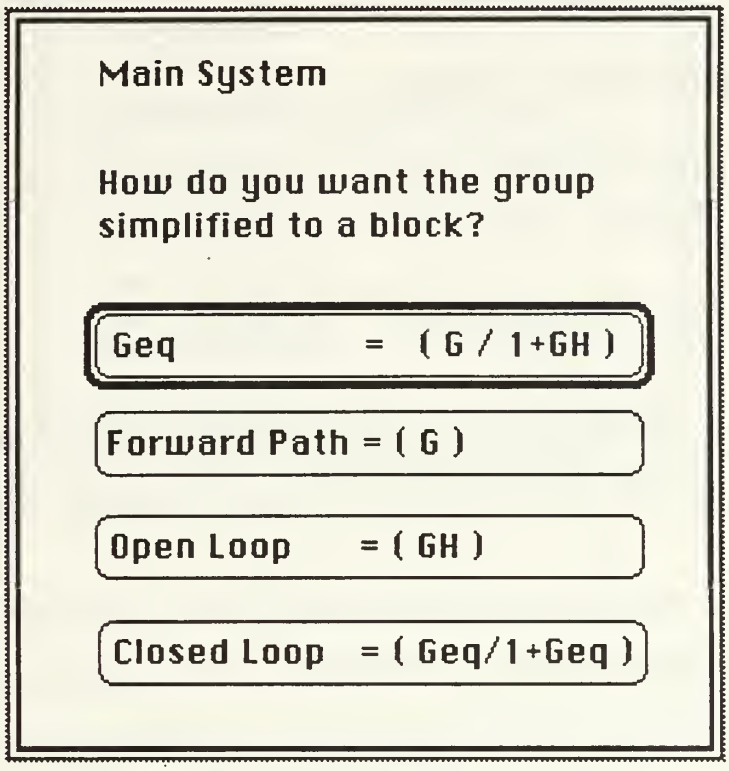


Fig(10) Denominator Data Entered.

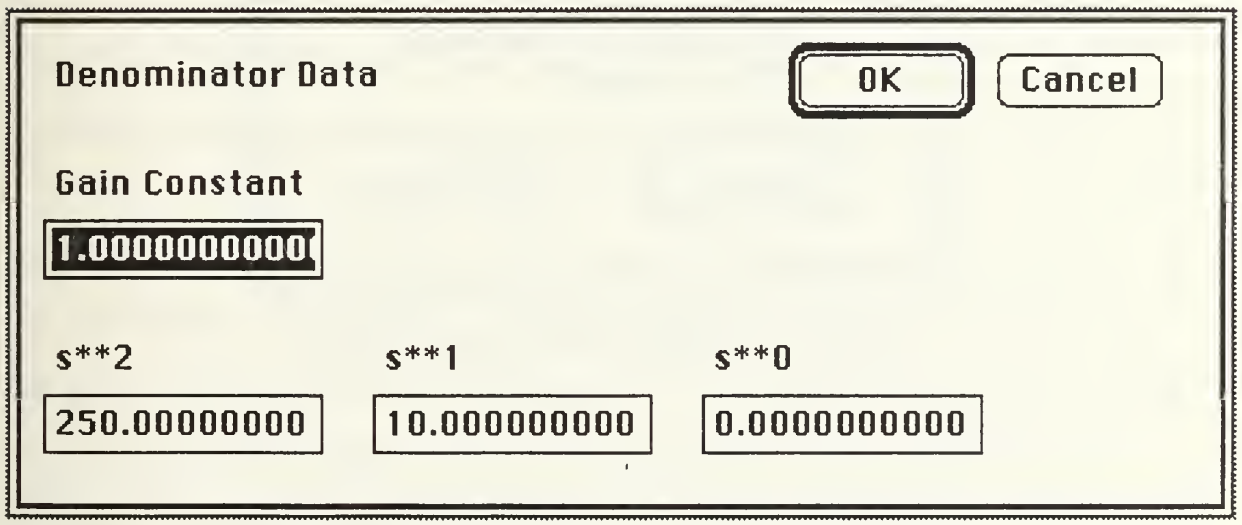
It should be noted that a Gain Constant is also input. This applies to a constant that is multiplied by the whole polynomial. In this case it is just 1. Since this was the first block entered, the loop path is also requested in Fig(11).

For the inner loop, Geq is needed so the default is selected. The next block in the inner loop is entered in the same way. In this block, the denominator is second order but it of type one. Note how the denominator values are entered in Fig(12). In this case, you could have divided the J value of 250 from the equation and entered the 250 as a gain constant. The s^2 coefficient would then be 1.0 and the s

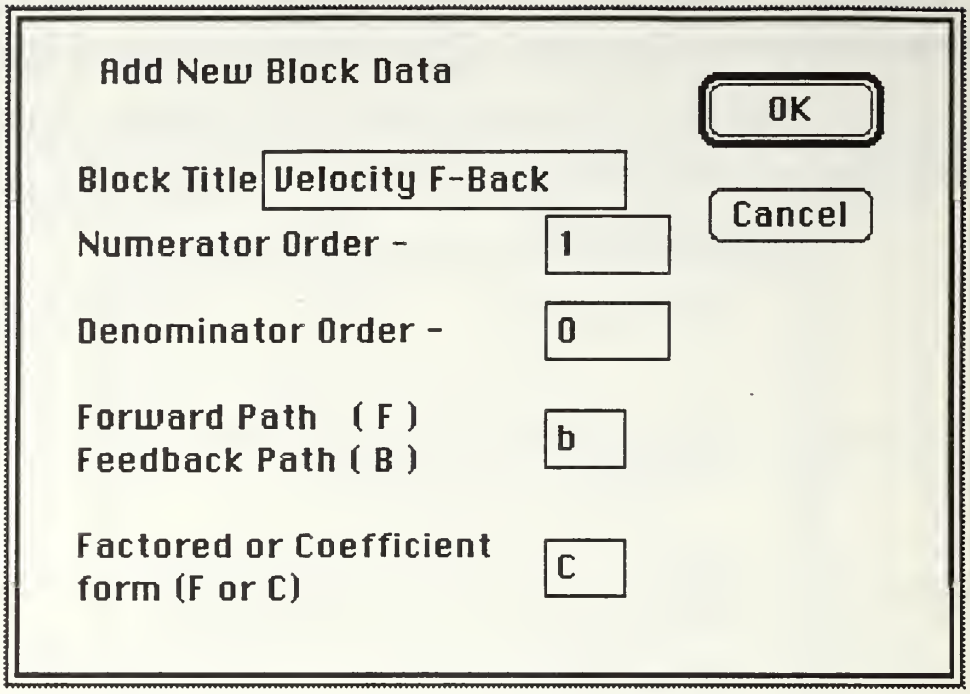
coefficient would be f/J or 0.04. The inner loop feedback block is now entered as in Fig(13) in the same manner as the others. In this case, the denominator order is 0 and the block is in the feedback path.



Fig(11) Requesting The Loop Path.

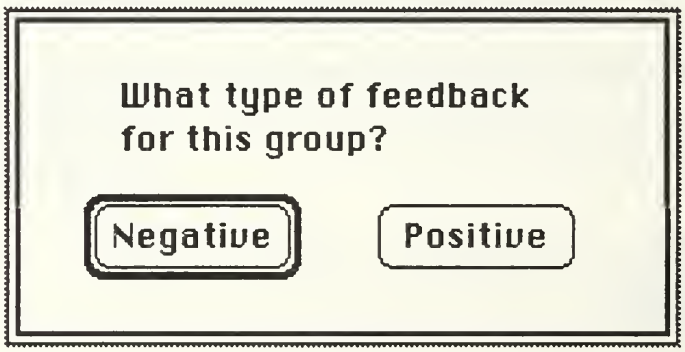


Fig(12) 'System Load' Block Denominator Data.



Fig(13) Entering The Velocity Feedback Block.

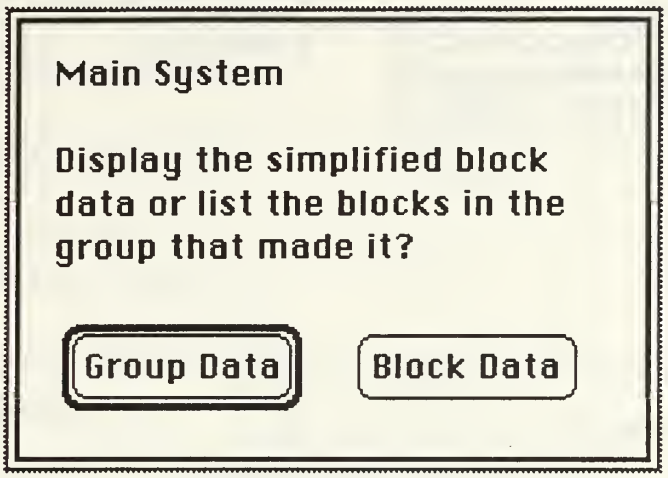
The data is entered in the same manner as the others but since this is the first back block in this group, the type of feedback must be determined. The default, Negative, is selected as in Fig(14).



Fig(14) Feedback Type Determination.

All the blocks in the inner loop have been entered and they can be viewed or changed as need be. To select a block to view, 'Change'

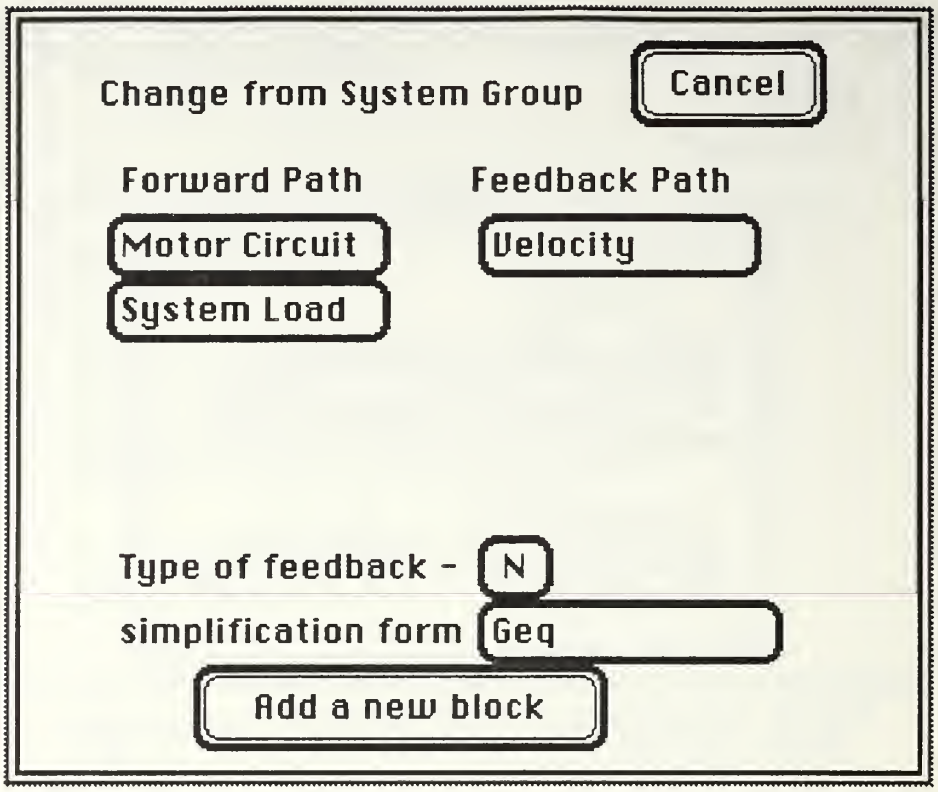
is selected from the Blocks menu. The group or block dialog box is shown in Fig(15).



Fig(15) Group Data or Simplified Block Data Selection.

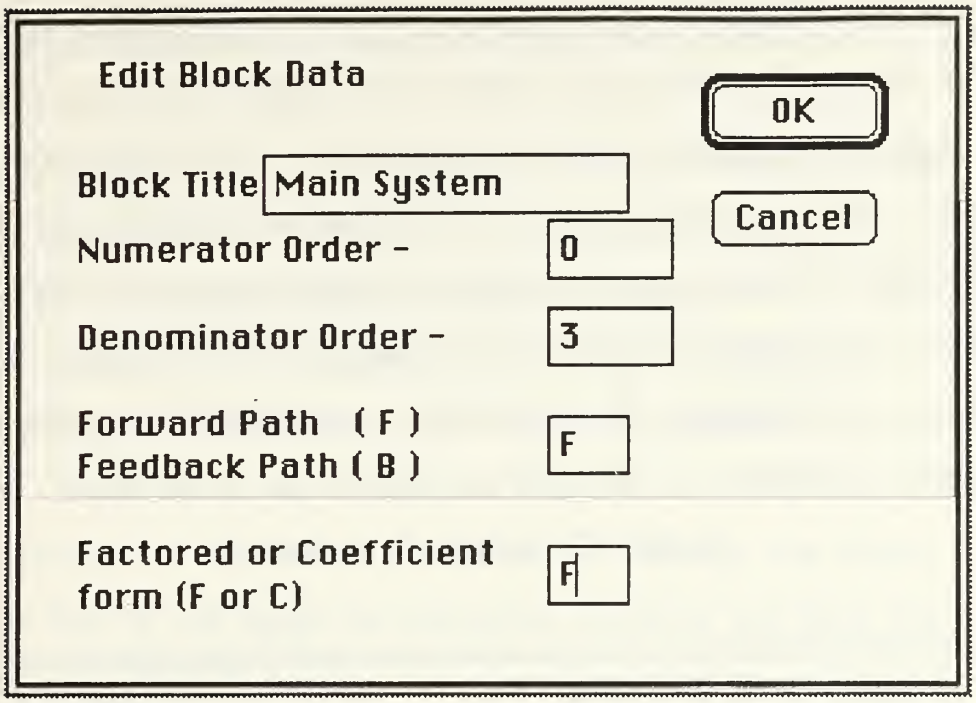
We want to look at the block that describes the motor characteristics so we select the default for Group Data. Fig(16) shows the dialog box which displays the blocks presently in the group.

The block titles are listed in two columns with the forward blocks in the left column and the back blocks on the right. The feedback type and loop path are also shown and can be changed by clicking on those buttons. A new block can also be added to the group by clicking on that button. The default is Cancel but we want to look at the Motor Circuit block. In the upper left corner, the title of the group we are looking at is displayed. In this case it is System Group.

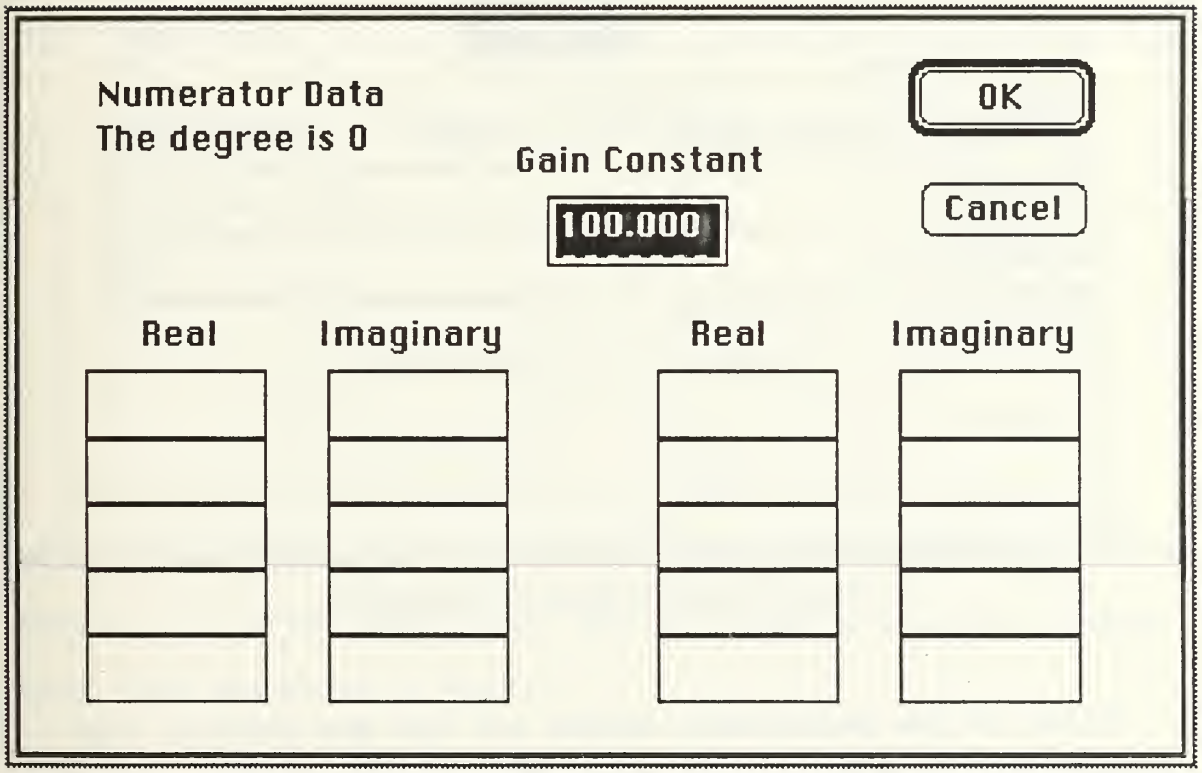


Fig(16) Selecting The Block To View.

Clicking on 'Motor Circuit' will display the same dialog boxes as Fig(8) through (10). Any of the data can be changed and saved if the OK button is pressed. We can also see the equivalent transfer function of all the blocks entered. This is done by selecting Change again but clicking on the Block Data button in Fig(15). The dialog box that describes the System Block is displayed in Fig(17). Note that the denominator order is shown as 3 which is as expected. This time we will view the polynomials in factored form. This will give us the roots of the characteristic equation, the denominator of the System Block. Fig(18) shows the numerator data. The numerator is zero order so there are no roots.



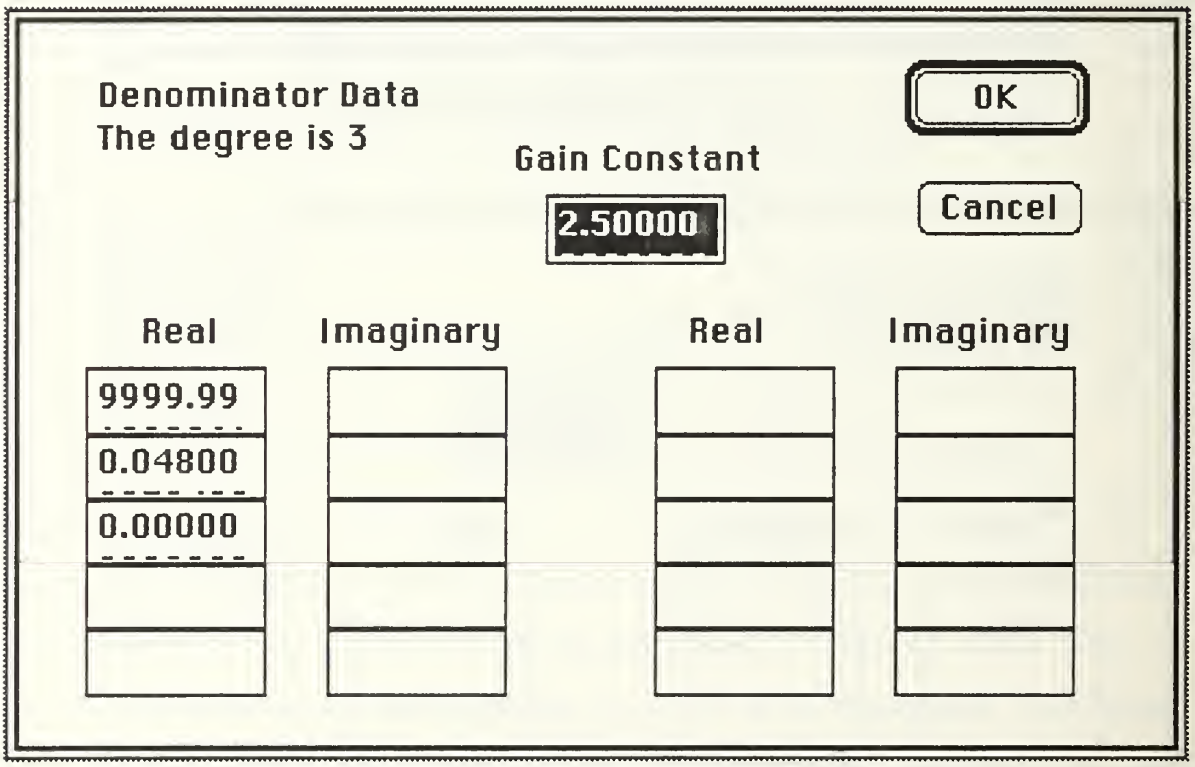
Fig(17) System Block Data.



Fig(18) System Block Numerator.

The denominator is third order so there should be three roots. Fig(19) shows this. Complex numbers are listed in two sets of real and imaginary columns. If there is a number in the real column but nothing in the imaginary column, then there is a single real root at that location. If there was a number in the imaginary column, then the real and imaginary numbers represent a complex pair in the form of Real +/- J Imaginary. These numbers might more accurately be described as factors. A factored polynomial is in the form;

(s+factor1) (s+factor2) (s+factor3)

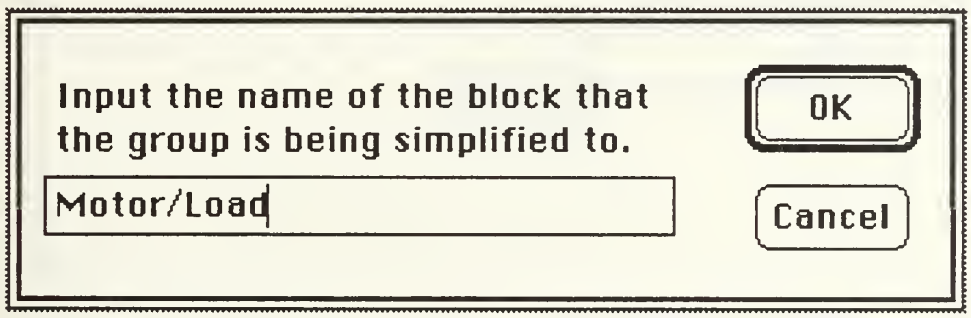


Fig(19) System Block Denominator.

Since all the denominator factors are real and positive, that means all the roots are real and negative which indicates that

there are no closed loop poles in the right hand plane for the inner loop.

We are ready to enter the outer loop. To do this, the inner loop must be simplified to one block. This is done by selecting Simplify from the Blocks menu. Fig(20) shows the request for the name of the new block. The three blocks input previously have been simplified to one block called Motor/Load. By selecting Change from the Blocks menu we would see that it is now the only block in System Group. If we select this block for viewing, the dialog shown in Fig(15) will again be presented because we have the option to view Motor/Load as one simplified block or as a group of blocks.

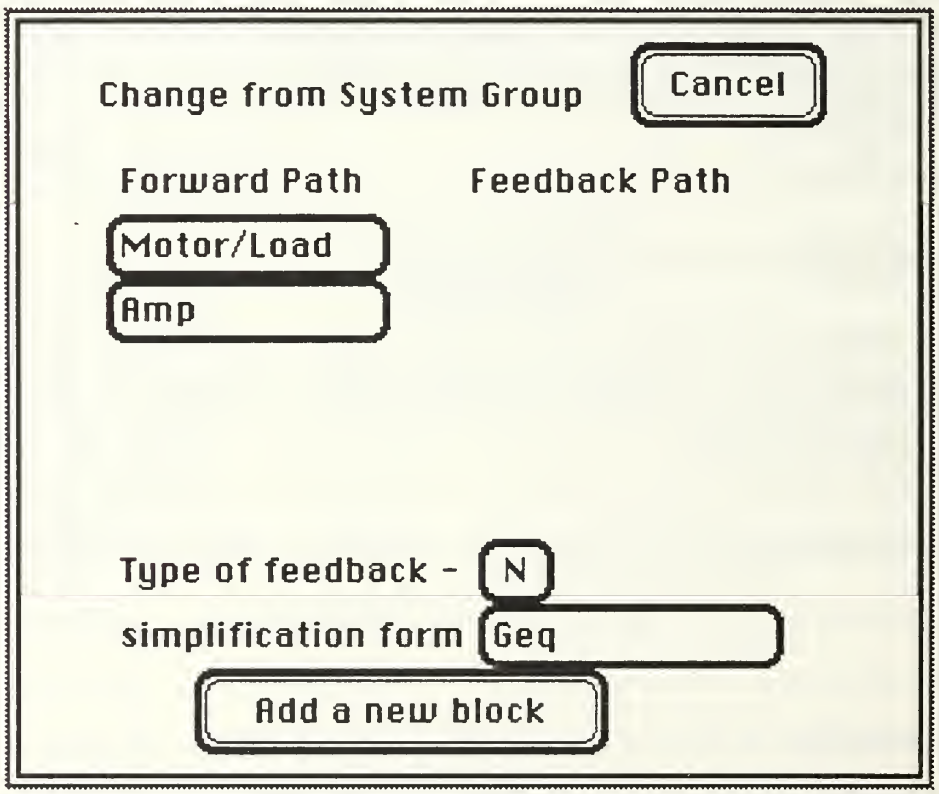


Fig(20) Simplifying The Group.

The amplifier will be entered as a forward block of zero order numerator and denominator. It will be entered the same as the others and it will be called 'Amp.' We can view the blocks in System Group. They are shown in Fig(21).

This is a good time to check to make sure the proper feedback and loop paths are set for the outer loop. Since there are no back blocks in the system Geq must be changed to 'Closed Loop' in order to have unity feedback. Clicking on 'Geq' displays the selection of loop paths allowing 'Closed Loop' to be selected.

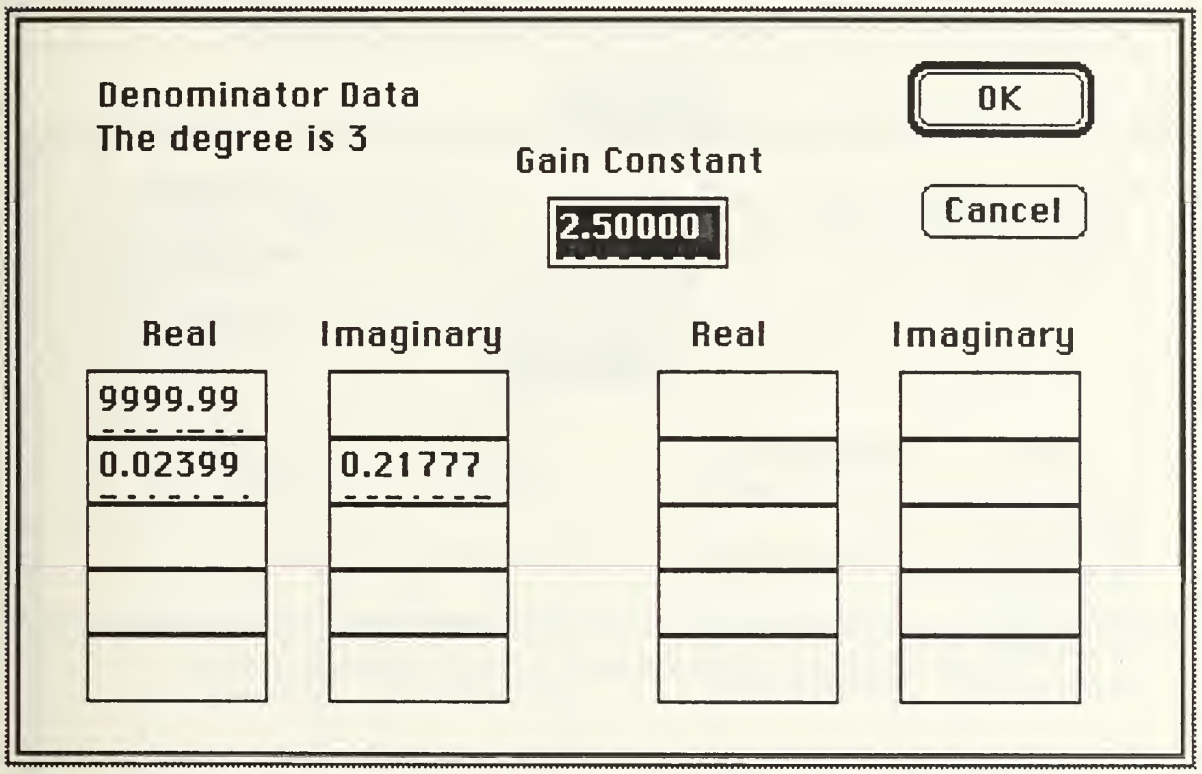
After checking the block values, we can see what the equivalent transfer function in System Block looks like. We will do this as done before and will observe the polynomials in factored form.



Fig(21) Viewing Complete System Group.

Fig(22) shows the denominator in factored form. The numerator was still zero order which is as expected with unity feedback and

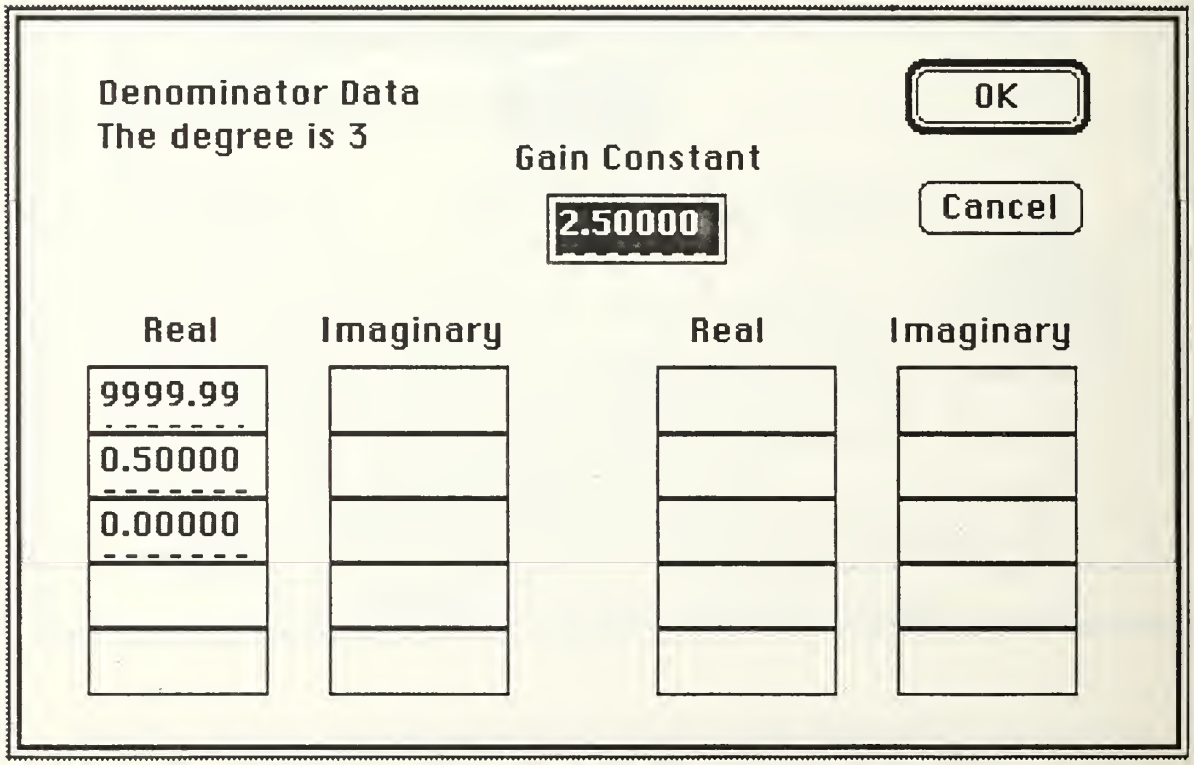
the denominator is third order. This time the characteristic equation has a pair of complex roots. They are all in the LHP but the real part of the complex pair is small so stability may be questionable. We can see how making a change to the Motor/Load block might affect the System Group roots. We can change Motor/Load as a single block rather than changing the blocks in it's group. This is done through the following chain of events. Select Change from the Blocks menu. Select Group Data for the blocks in System Group. Select the Motor/Load block. This time select Block Data so Motor/Load will be displayed as the equivalent block from the earlier simplification.



Fig(22) Denominator Of System Block.

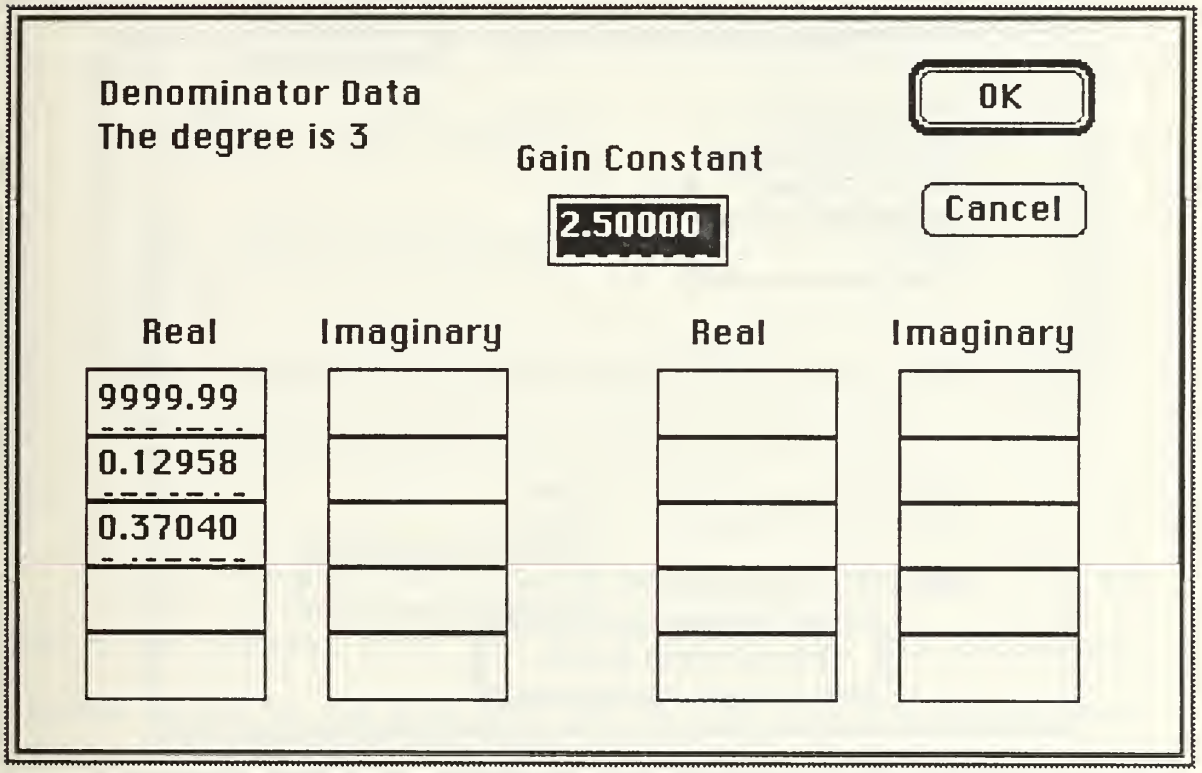
Enter 'F' for Factored or Coefficient Form when displaying the block data as in Fig(8). Hit return when the numerator is displayed in factored form because we want to change the denominator. Fig(19) will again be displayed showing the factors. Change the .048 to .5 by double clicking in it's box and typing in the new number. It will now look like Fig(23).

By clicking on OK, the new roots are saved but the original blocks that make up Motor/Load have not been changed. This means we can return it to it's original values. Select Change again from the Blocks menu but select Block Data in order to observe the equivalent system transfer function. Enter 'F' for factored form and see what changes may have occurred to the characteristic equation.



Fig(23) Adjusted Motor/Load Denominator.

Fig(24) shows the new denominator roots are all real and in the left plane. These roots would appear to give a more stable system so the changes made to Motor/Load were good. Now we must determine what changes must be made to the blocks that make up Motor/Load. This can be accomplished by using the Root Finder under the Tools menu but that will be discussed in a later chapter. Motor/Load must now be changed back so it accurately reflects the values from it's blocks.

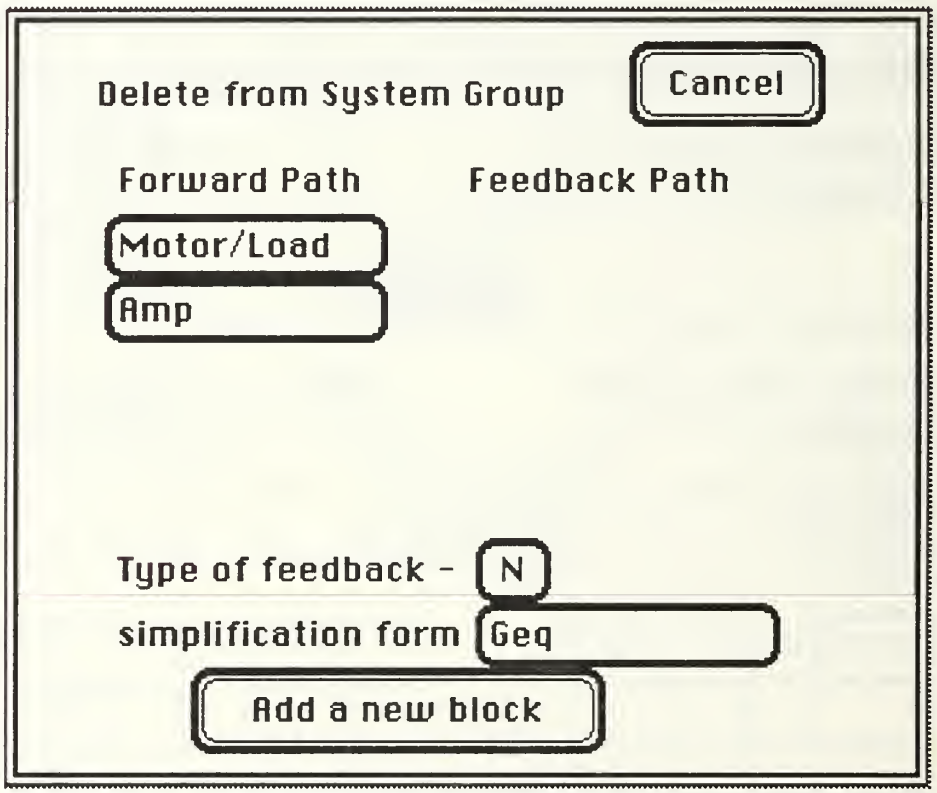


Fig(24) Adjusted System Characteristic Equation Roots.

This is done by viewing any of it blocks and hitting 'enter' or 'return' on the keyboard as it is being displayed. It should be displayed in coefficient form so no changes are actually taking

place. OK is selected however, to tell MacCAD to recalculate the values for Motor/Load and System Block.

Deleting a block is done much the same as viewing a blocks contents. If we wanted to see how the system would work if there was no amplification of the error signal, we could do this by deleting the Amp block. Select Delete Block from the Blocks menu. Fig(25) shows the dialog box showing the blocks that can be deleted.



Fig(25) Deleting Block From System Group.

The dialog is the same format as the 'Change' dialog except it says 'Delete' in the upper left corner. You can also make the feedback and loop path changes or even add a new block as you can in the

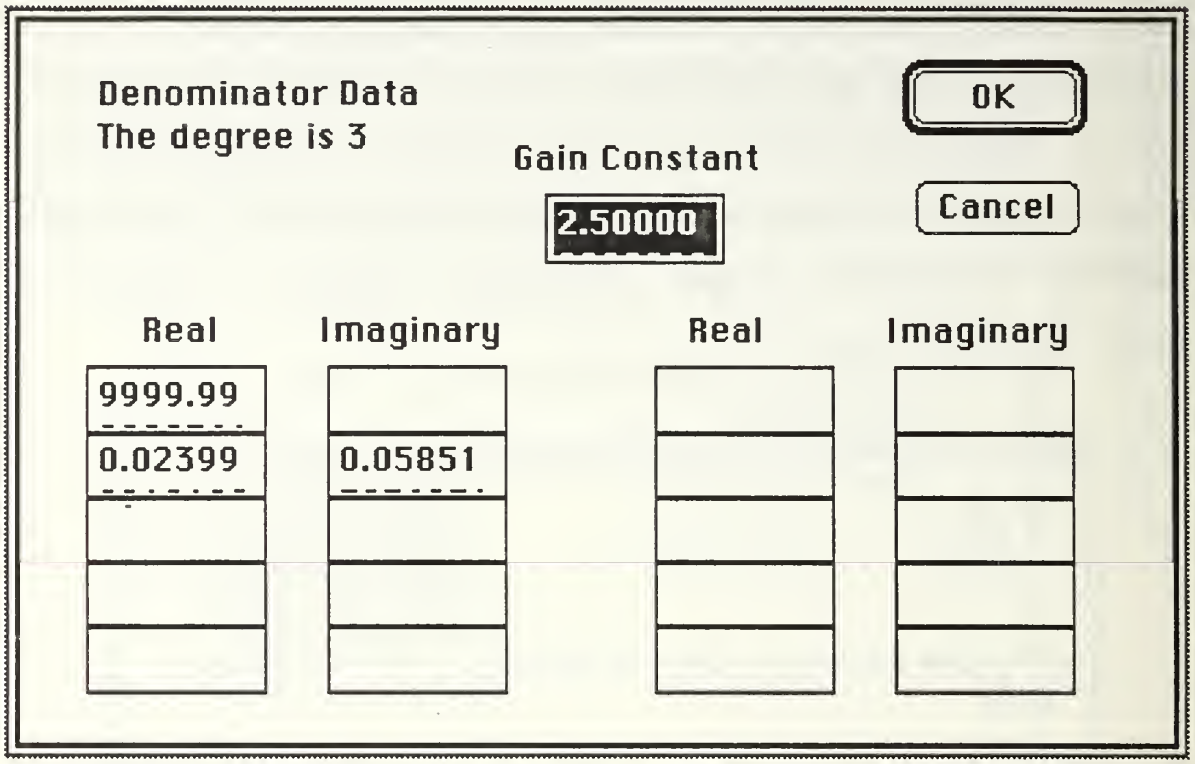
Change dialog. If you do not notice the word 'Delete' in the left corner and think you are going to view the contents of a block and select a block that you did not want do delete, you will be questioned by the alert box shown in Fig(26). In this case, we do want to delete 'Amp' so the 'Yes. Do it.' button is selected. Selecting Change again and viewing the Group Data shows that 'Amp' has been removed. The resulting system characteristic equation is checked and is shown in Fig(27).

Fig(26) Safeguard Preventing Inadvertent Deletions.

We can see that removing the amplifier did not affect the real part of the complex roots, just the imaginary part was decreased. Various values for the amp gain can be tried and it is quickly seen that increasing the gain increases the imaginary part of the complex roots. Such trial and error calculations can be completed quickly.

In conclusion, all the system data is entered in as blocks. Groups of the blocks can be simplified to a single block but the original blocks are still in the system. They can be changed or deleted as

need be. Feedback types, loop paths and gain constants can be easily changed and the results observed.



Fig(27) Characteristic Equation Roots Without 'Amp'

V. BODE PLOT

A. BASIC DESCRIPTION.

The Bode Plot is probably the most useful, as well as most used, single tool for system analysis. It displays graphically, the system output magnitude and time delay as functions of the frequency of the input. The output is the steady state response of the system due to a constant amplitude sine wave of a given frequency. Magnitude is plotted in decibels, or dBs from the equation:

$$\text{Magnitude (dB)} = 20 \times \text{Log(output/input)}$$

Time delay is the difference between the positive zero crossing of the input signal and the positive zero crossing of the output signal. It is shown as phase in degrees with 360 degrees representing the input signal's period. Since the steady state output will always be of the same frequency as the input, their periods will also be equal. If the output lags the input by an amount of time equal to 1/4 the input signal's period, the phase will be 90 degrees, being 1/4 of 360 degrees. A time delay greater than the signal's period would still be shown as some angle between 0 and 360 degrees since both signals are periodic and when dealing with steady state responses, it is impossible to determine which individual input zero crossing actually caused any specific output zero crossing.

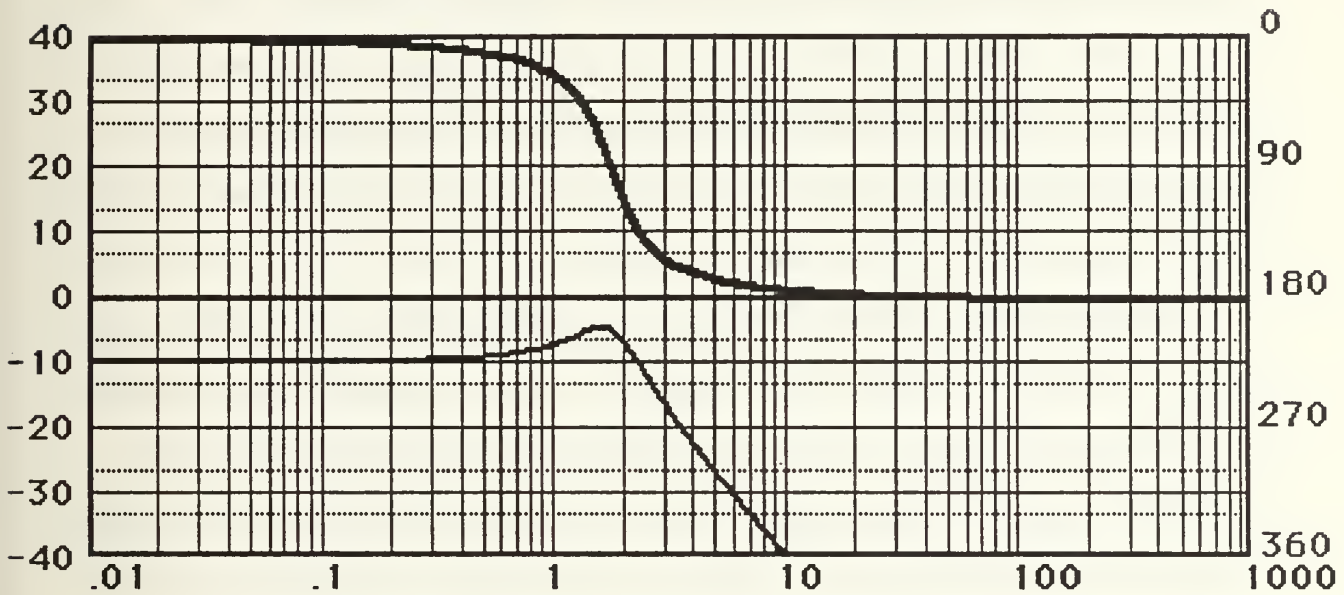
The Bode plot offers a wide variety of information to the user. At a glance, it shows what frequency range will be passed or attenuated by the given system. As in the case of a band pass filter, it will show the pass band, the transition band, the corner frequencies as well as the respective gains. The phase plot also gives information regarding the phase margin and gain margin and how stable the system is. With the basic knowledge of Bode plot characteristics such as 20 dB/decade slope of the magnitude curve and 45 degrees/decade slope for the phase curve from a single pole or zero allows for easy approximations for compensators needed to get the system to respond in the desired manner.

B. USER OPTIONS.

After transfer function data has been entered into the System block, you are ready to calculate and display the Bode plot. Before actually plotting the Bode data, you select from several options. A brief description of the options follow. An example at the end of this chapter will walk you through the use of the Bode plot tool.

You select the range of the magnitudes to be displayed by inputting integers for the lowest and highest magnitude, in dBs. The magnitude scale will appear on the left vertical axis of the plot. There will be 6 or more divisions between the upper and lower magnitude limits. The divisions will be calculated so as to always evenly fall on whole numbers. Regardless of how the divisions are displayed, a zero magnitude line will always be included. The solid

horizontal lines correspond to the magnitude values on the left of the plot. The dotted lines are for the phase curve. Fig(1) shows a Bode plot for a simple system. The phase range of the plot will always be from 0, at the top, to 360 at the bottom. These numbers refer to the delay of the output compared to the input. The numbers cannot be changed. As mentioned earlier, the dotted horizontal lines correspond to the phase values on the right of the plot.



Fig(1) Example Bode Plot Of Simple System.

The frequency of the input signal is shown on the horizontal axis which is in a logarithmic scale. The solid vertical lines refer to the frequency values. MacCAD allows the user to select the lowest and highest frequency to display. This information is entered as integers in the form of powers of 10. The thick line on the plot is the phase curve and the thin line is the magnitude curve. You may also select

not to display the phase curve. This can be of use particularly when you want to display several curves at the same time and the magnitude is of prime concern.

When 'Bode Plot' is selected from the 'Tools' menu, you also have the options to cancel the operation, redraw the last plot displayed, draw a new plot from scratch or to overlay a new plot on the last plot. In addition, you may add title boxes anywhere in the Bode plot window. You will be asked what text to display and the box size is automatically calculated based on the number of lines you enter, and the length of the longest line. You can add as many titles, or labels as you want and you can put them anywhere. They will be drawn on top of the Bode plot so they can be placed over a part of the plot grid that is not used by the curves.

Multiple plots can be displayed by selecting 'Overlap Plots'. This plots the current System block transfer function Bode plot on top of the last plot displayed. In order to align the plots, the same phase, magnitude and frequency limits are used as the last plot. The plots can be differentiated by the pattern of the lines. As more plots are overlapped, the lines become lighter so each pair of curves can still be identified. As with any other window displayed on the Macintosh, the Bode plot window can be moved, resized, scrolled, closed or opened with a single mouse movement or a few key strokes.

C. PROGRAMMER'S NOTE. BODE DATA CALCULATION AND DISPLAY.

When the 'Bode Plot' tool is selected from the menu, the program enters the 'Bode' unit and the 'DoBodeMenu' procedure until the entire

plot has been calculated and the display is complete. The empty grid is drawn using the 'DrawBasicPlot' and 'LabelPhase' procedures which use data from the global 'BodeData' which contains the following fields.

```
BodeData = record;
     minfreq : integer;
     maxfreq : integer;
     minmag : integer;
     maxmag : integer;
     layer : integer;
     doit : boolean;
end;
```

The magnitude and frequency have already been explained. The 'layer' field shows the number of plots that have been displayed by 'Overlap Plot' and 'doit' is a boolean flag to show that through all the dialog boxes requesting data from the user, he has not selected cancel at any time.

When 'Bode Plot' is first selected, the user has the option to redraw the last plot, draw a new plot, overlap plots or cancel the operation. Redrawing the last plot is done by simply opening the Bode window. Drawing a new plot is done by getting plot data from the user, calculating the points and drawing the plot. Overlapping plots is done by opening a temporary picture. The last window picture is first drawn into it followed by the new magnitude and phase curves which are also drawn into it using Quickdraw commands. The temporary picture is closed and then set to be the window picture for the Bode window.

During program startup, the BodeData variables are initialized to default values that are most likely to be used. Whenever the user

to default values that are most likely to be used. Whenever the user selects some other values for a display, they become the new default values for the next time. The 'GetBodeData' procedure displays the dialog box that asks the user for these values. The default values are loaded into the data boxes by the 'InitBodeData' procedure. Any inputs from the user are checked to ensure they are integers and that they make sense. For example, if the maximum magnitude was less than the minimum magnitude, an error would be flagged using the 'GoodBodeDataEntered' procedure.

Once all the inputs from the user are completed, the plot must be drawn. Horizontal and vertical positions on the plot are calculated using functions that convert between frequency (for horizontal) or magnitude (for vertical) values and actual pixel numbers. The functions also use the global BodeData information. These functions, Freq2Wd, Wd2Freq and Mag2Ht draw all the horizontal and vertical lines used for the entire plot. The actual curves are drawn by calculating a single point for the magnitude and phase based on a frequency value in the procedure 'PlotMag'. The actual operation of 'PlotMag' will be discussed in another chapter. After each point location is determined, it is checked to see if it lies within the plot boundaries. If it does, then a line is drawn from the last point to the current one. Frequencies are selected to ensure the horizontal length of any line drawn will not be greater than 3 pixels. This makes the overall curve actually look like a curve, rather than a series of connected lines. In order to ensure that lines such as the thick phase curve drawn close to the border of the plot

do not go outside the boundaries, the window clipping region is set to 'plotrect' which is the rectangle that defines the plot shape. After the curves have been drawn, the clipping region is returned to the full screen size to ensure proper scrolling and window sizing.

Titles are added by using the 'AddLabel' tool. This tool can be used on any window containing a plot drawn by MacCAD. The user enters text up to 255 characters per line and up to three lines. This is done through a dialog box called by 'GetLabelData'. The size of the label box is determined by the number of lines used and the length of the longest line of text entered in pixels. After the text is entered and the size of the label rectangle is determined, the 'DoLabelMenu' procedure waits in a 'while' loop for the user to press the mouse button. When 'button' is true, a rectangle is drawn using the PatXor penmode, with the mouse position being the top left corner. When the mouse is moved, the last drawn rectangle is drawn again which due to the PatXor mode, actually erases the last rectangle and returns the display to what ever was under it before. A new rectangle is also drawn. This is continued for as long as the button is down. As soon as it is released, the last rectangle is erased in the above manner and the actual label is drawn into a picture called 'labelPic' using the 'DrawLabel' procedure. The user is asked if he wants to save the label as shown by an alert box. If the default Yes is selected, then the label picture is added to the original window picture with the 'AddPic' function from the Programmer's Extender library.[1] Because of the way pictures are

[1] This library is described in the appendix.

added or drawn together, as many labels can be made as desired, just as with 'Overlap Plots'.

D. BAND PASS FILTER EXAMPLE.

In this example a band pass filter will be designed with a center frequency at 10 Hz and a gain at that frequency of 20 dB. The only other requirement on the filter is that a 1 dB pass band have a band width of as close to 5 Hz as possible. The overlap capability will be used to compare the magnitude frequency response after selecting different transfer function values.

The second order band pass filter transfer function will be in the form of:

$$H(s) = \frac{K(wc/Q)s}{s^2 + (wc/Q)s + wc^2}$$

Where ωc is the center frequency, Q is the quality factor and K determines the gain at ωc. Q is used to describe the 'sharpness' of the bandpass or the bandwidth. The bandwidth is the distance between the half power points, or the points that are 3 dB below the gain at wc. The bandwidth is related to wc and Q by the equation:

$$\text{bandwidth} = \omega c/Q$$

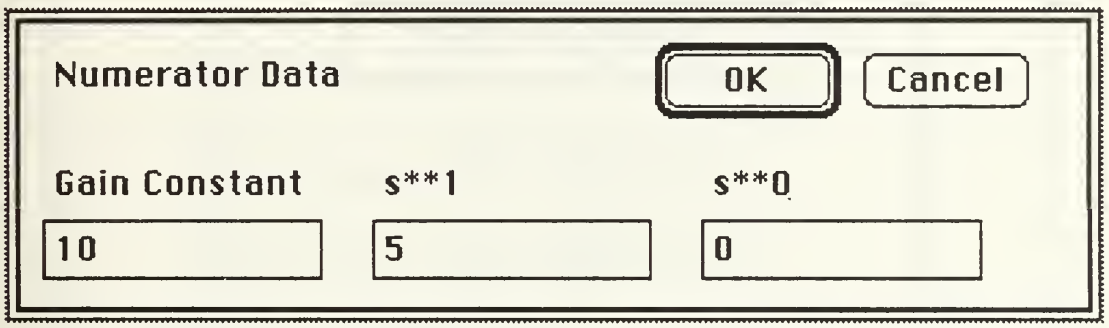
When ωc = 20, K can be calculated by;

$$\text{gain(dB)} = 20\ \text{Log}(K) = 20\text{dB} \quad \text{or} \quad 10 = K$$

The only variable to adjust is Q. Since we know that the half power bandwidth will be greater than the 1 dB bandwidth, Q can not be larger that 2. We will start with this value, check the Bode plot and then make adjustments as necessary.

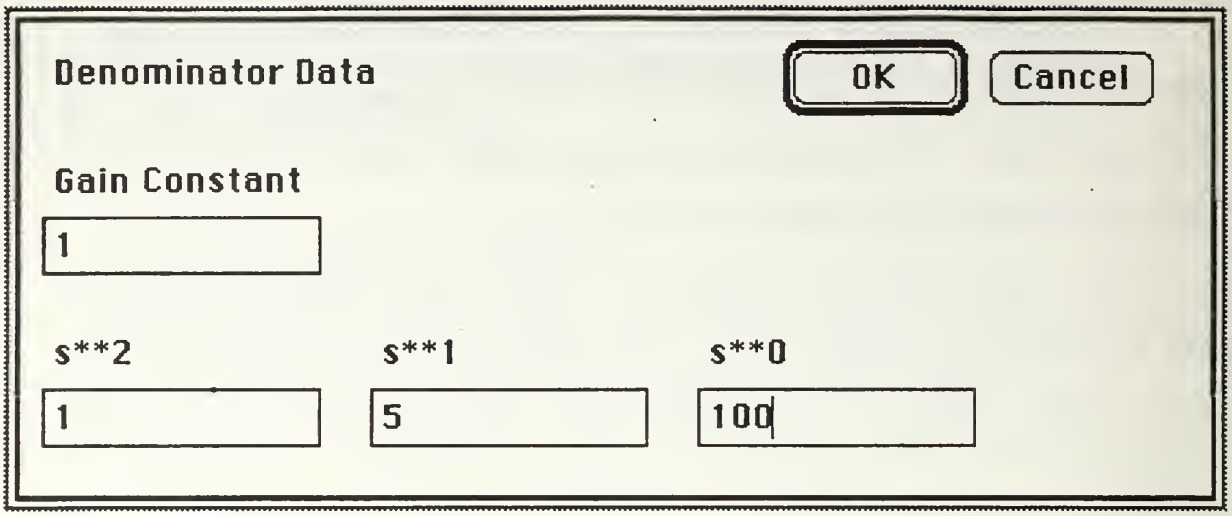
The system will be entered as a single block with the loop path set to the default 'Geq' which will not add the feed back since there are no back blocks in the System group. Fig(2) shows the numerator data input.

This is entered as the gain constant. Eq1 shows the numerator consists only of an 's' term with a coefficient = $\omega c/Q = 10/2 = 5$. This is input along with 0 for the s^0 term.

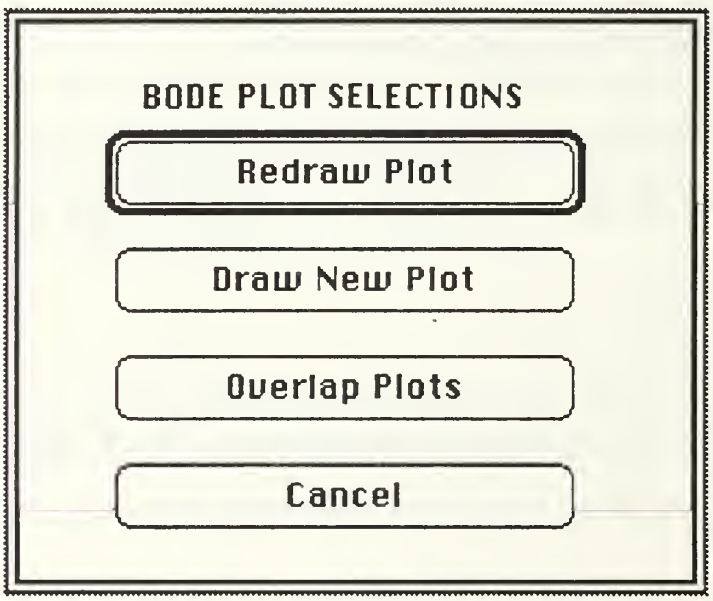


Fig(2) Band Pass Filter Numerator With Q = 2.0

Fig(3) shows the data input for the denominator. Since there is no gain constant in the denominator of eq1, 1.0 is entered in the gain constant data box. The coefficient of the s^2 term is also 1.0. The 's' coefficient is $\omega c/Q = 10/5 = 2$. The last term is $\omega c^2 = 100$. The loop path is left to be 'Geq' again since there are no back blocks. 'Bode Plot' is now selected from the 'Tools' menu. Fig(4) shows the dialog box presented at this time.



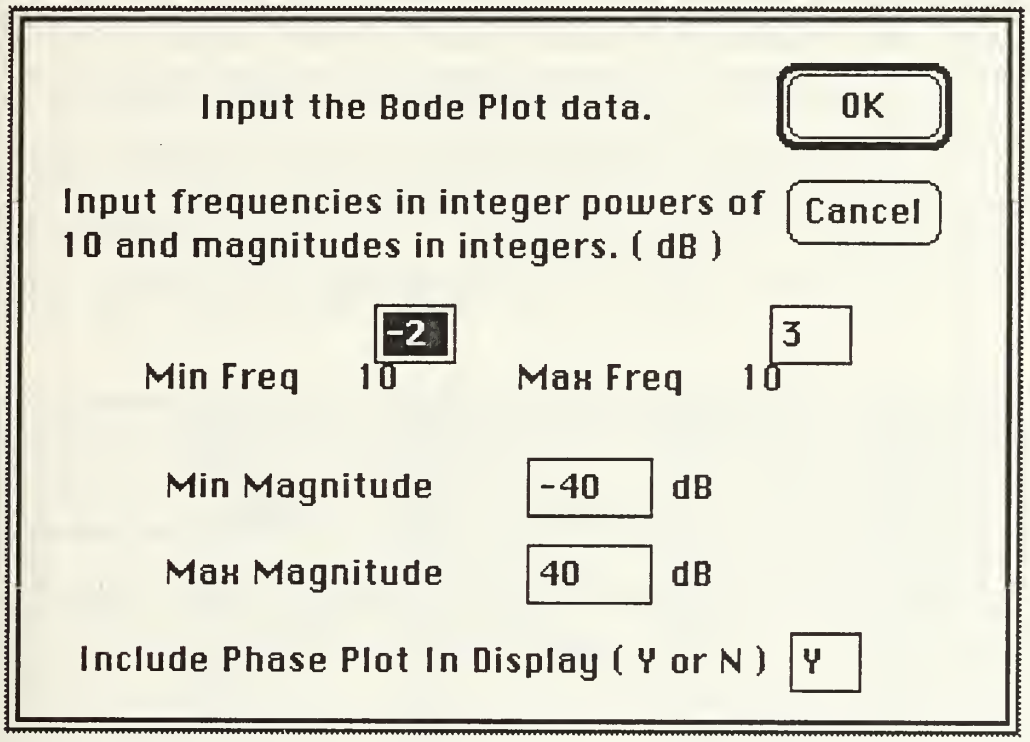
Fig(3) Band Pass Filter Denominator With Q = 2.0



Fig(4) Bode Plot Tool Dialog Box.

This lets you select one of the options described earlier. Since in most cases, you will wish to view the last plot drawn, 'Redraw' is the default but if no plots have yet been drawn, selecting 'Redraw Plot' would result in an alert box indicating that a plot has not yet

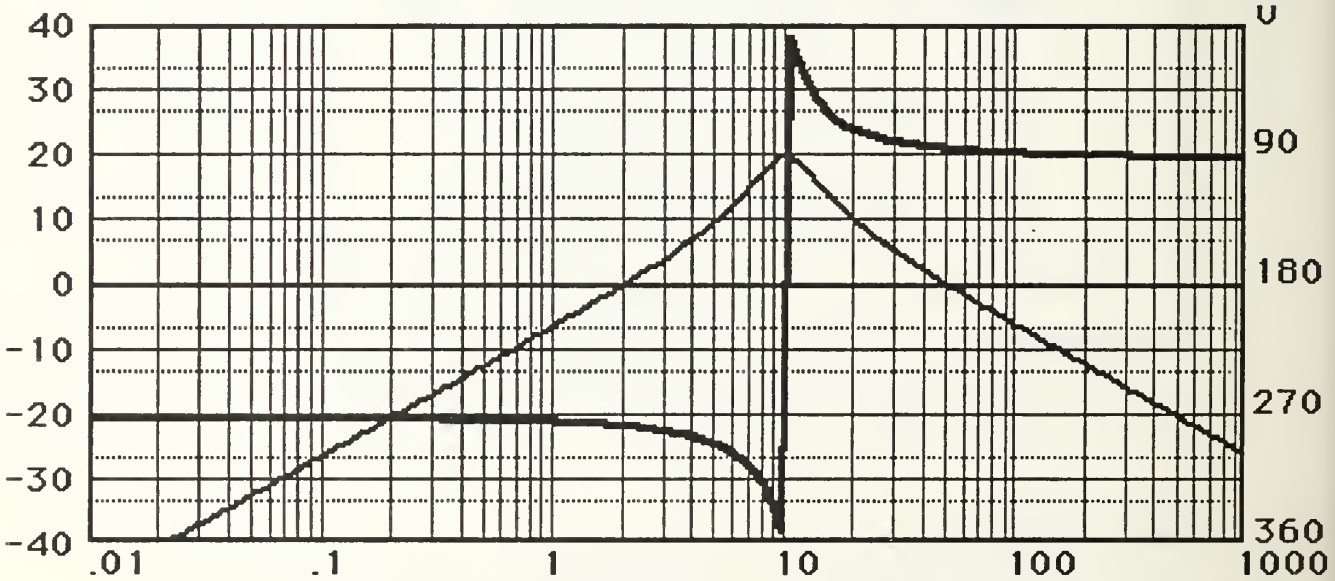
been drawn. 'Overlap Plots' would do nothing for the same reason. In this case we want to 'Draw New Plot.' The dialog box that inputs the Bode data is shown in Fig(5). In our case, the default data should work fine. We could change the magnitude values or the frequency range or even prevent the phase curve from being drawn but all these options will be used later. Right now we accept the default data.


Input the Bode Plot data.
OK
Input frequencies in integer powers of 10 and magnitudes in integers. (dB)
Cancel
Min Freq 10 -2
Max Freq 10 3
Min Magnitude -40 dB
Max Magnitude 40 dB
Include Phase Plot In Display (Y or N) Y

Fig(5) Bode Data Dialog Box With Default Data.

After clicking 'OK' a notice is displayed on the screen stating that the Bode data points are being calculated and we should be patient. A counter is also shown that counts down the data points to be calculated. After reaching zero, the Bode plot is drawn as in Fig(6). Something interesting has occurred with the phase curve. It

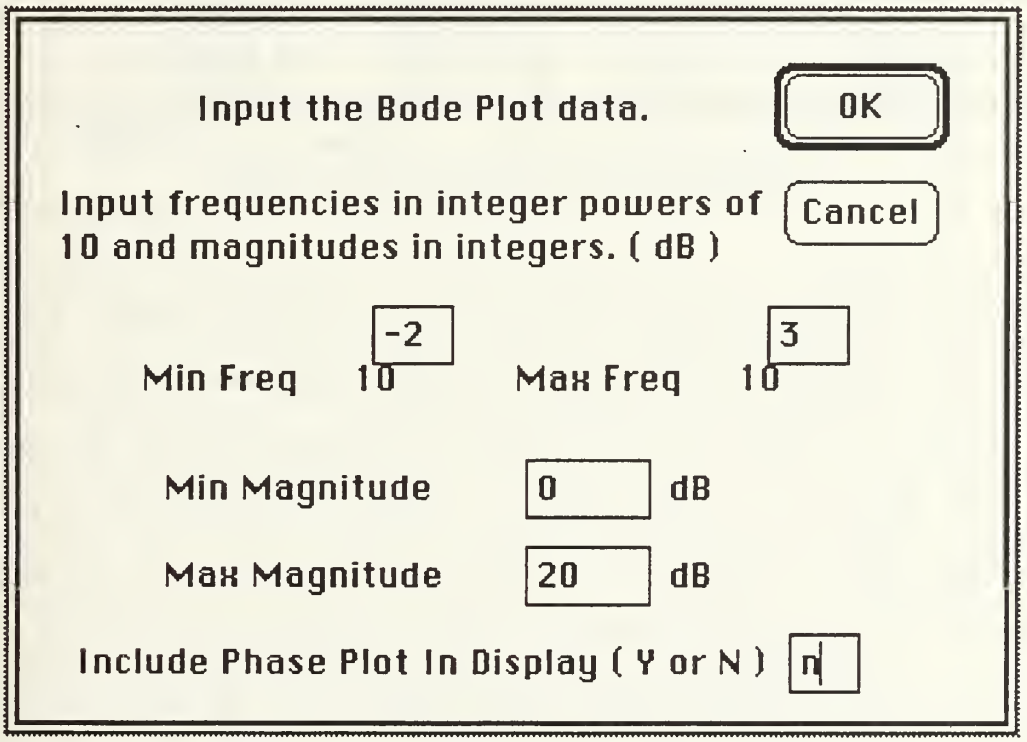
started at the bottom of the plot and then suddenly jumps to the top. At first this could be confusing but it is correct. Since the phase is plotted from 0 to 360 degrees, a value of -90 would be plotted as +270. The phase plot starts at the left at a value of 270 which is the same as -90 or means the output leads the input by 90 degrees. This makes sense since the numerator has a single 's' term, or in other words, there is a 'zero' at 0 where 'zero' means a root of the numerator. Since the denominator has a nonzero s^0 term, we know that there is not a pole at 0 to compensate the 'zero' there. As frequency increases, the phase decreases to a lag situation. When the phase moves from 359, a one degree lead, to 1, a one degree lag, the phase plot moves abruptly from the bottom of the plot to the top.



Fig(6) Original Band Pass Bode Plot.

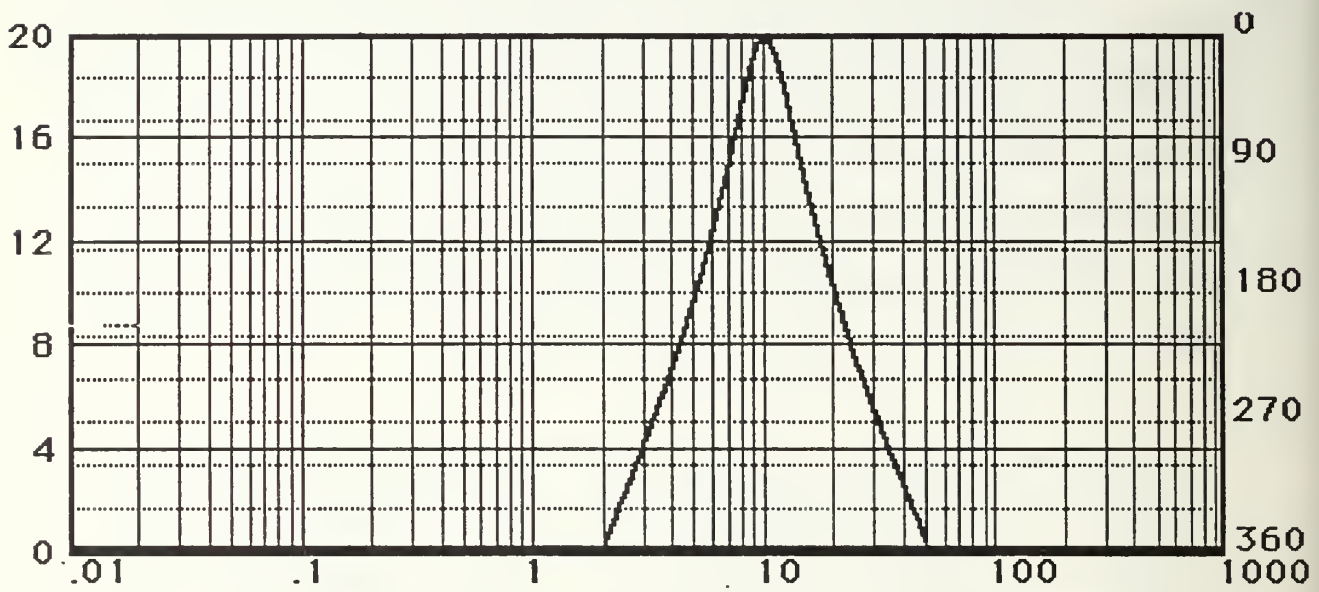
Since the denominator is second order, we know that the system will always be stable, regardless of the value of Q selected. If Q is

very small, the 's' term will be very large and, as can be shown using the 'Root Finder' tool, the real part of the complex roots, if they are complex, will always be in the left hand plane. As Q approaches infinity, the real part of the complex pair will approach zero from the positive side, meaning the roots will again be in the left hand plane. The rule of thumb is that for quadratics, if all the terms are greater than zero, then all the roots will be in the left hand plane. Since we no longer have to worry about the stability, the phase plot is not of great concern to us. We can elect not to plot it. As for the magnitude curve, it hit a peak at 20 dB at ωc as expected but the plot is too large to see the 1 dB bandwidth. We will draw a new plot with the plot data changed as shown in Fig(7).


Input the Bode Plot data.
OK
Input frequencies in integer powers of 10 and magnitudes in integers. (dB)
Cancel
Min Freq 10 -2
Max Freq 10 3
Min Magnitude 0 dB
Max Magnitude 20 dB
Include Phase Plot In Display (Y or N) n

Fig(7) Adjusted Bode Plot Data

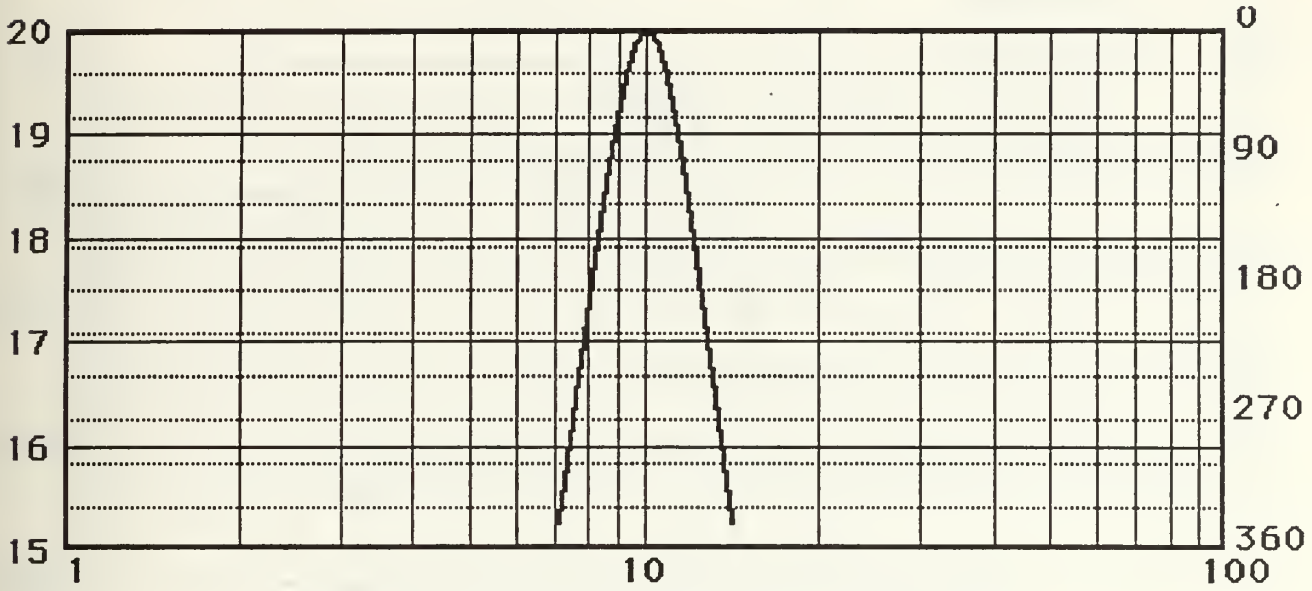
The frequency limits are not changed but the gain is changed to go from 0 to 20. This should better show the 1 dB bandwidth. It was also decided not to display the phase curve so 'N' was entered in the last data box. After 'OK' is selected and the 'Please be patient.' box has disappeared, the new Bode plot looks like Fig(8).



Fig(8) Plot From Adjusted Bode Data.

With the phase curve removed, the plot is much clearer and the magnitude change also helped but it appears that we need to zoom in even more. This time we will change the frequency limits to cover a frequency range of 1 to 100, which in powers of 10 would mean we input the numbers 0 and 2 for the min and max frequency. We can also adjust the magnitude limits to have a maximum of 20 and a minimum of 15. This will clearly show both the 1 dB band width as well as the half power band width.

'Bode Plot' is again selected from the 'Tools' menu and 'Draw New Plot' is also selected. The new bode data dialog box is shown in Fig(9).

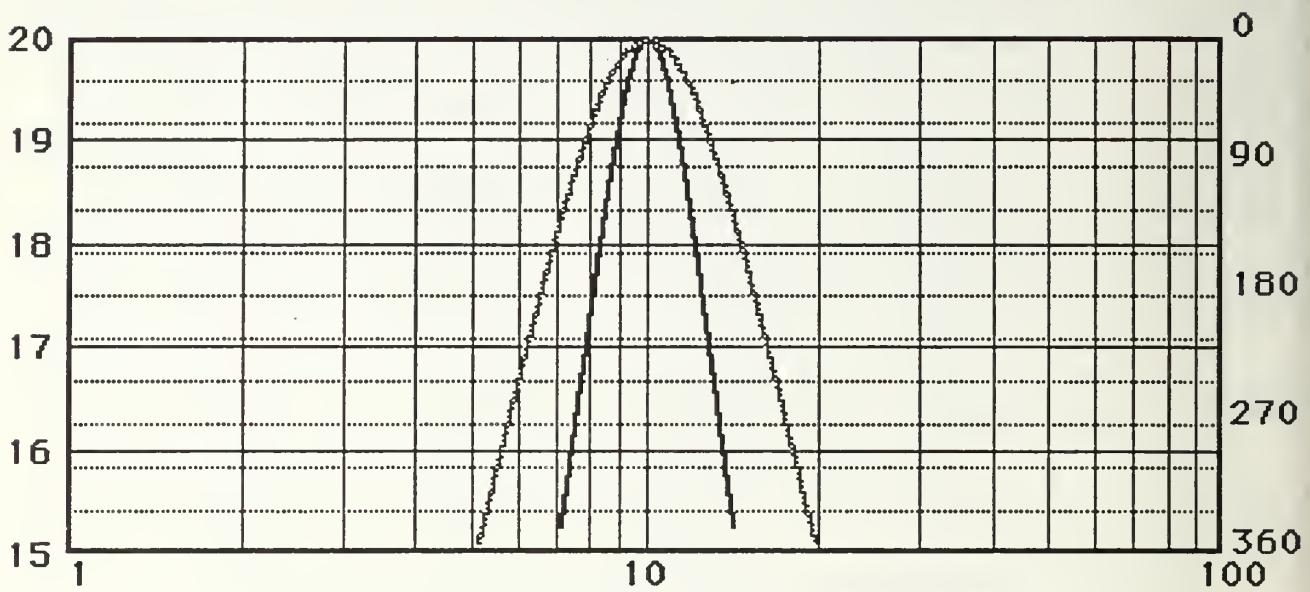


Fig(9) Bode Plot After Second Data Adjustment.

This plot clearly shows the 1 dB band width as well as the half power band width. The 1 dB band width is much less than 5 Hz as it is from about 9 Hz to about 11. This means we must decrease Q. We can try Q = 1.0. This means that both the numerator and denominator 's' terms must now be 10.0 instead of 5.0 as before. The changes are made to the system as described in the chapter covering the Block Manipulator. This time since the Bode data displays the plot well, we can 'Overlap Plots' in order to see how the changes in Q affect the band widths. Fig(10) shows the plot after the overlapping.

The plot now shows magnitude curves for Q = 2.0 and 1.0 with the lighter line being the later. The 1 dB band width now appears to

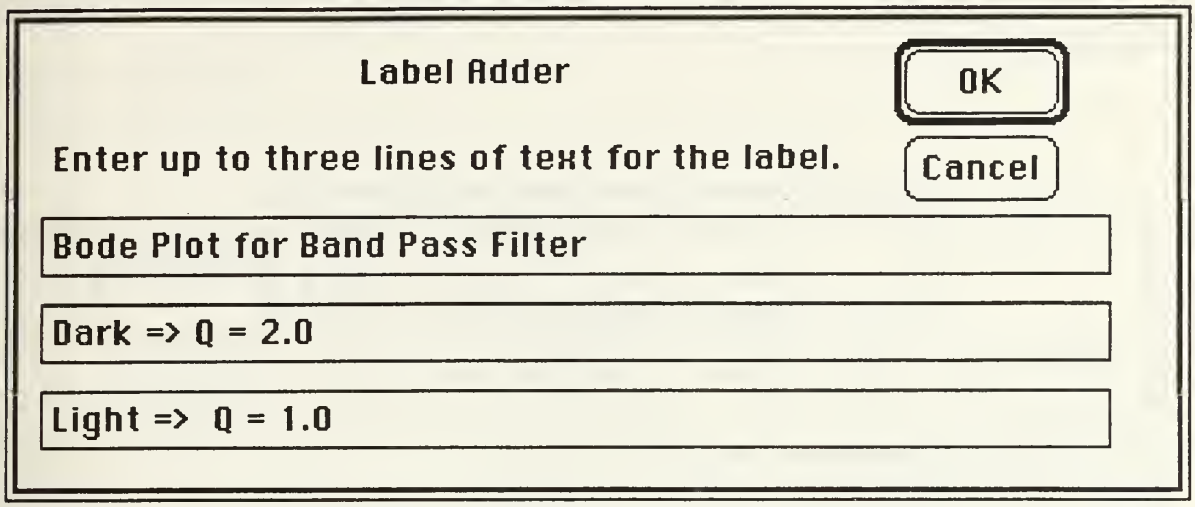
extend from 8 Hz to about 13 Hz which is the required 5 Hz band width. Now that we have the desired response displayed in the Bode plot, we can add labels for identification.



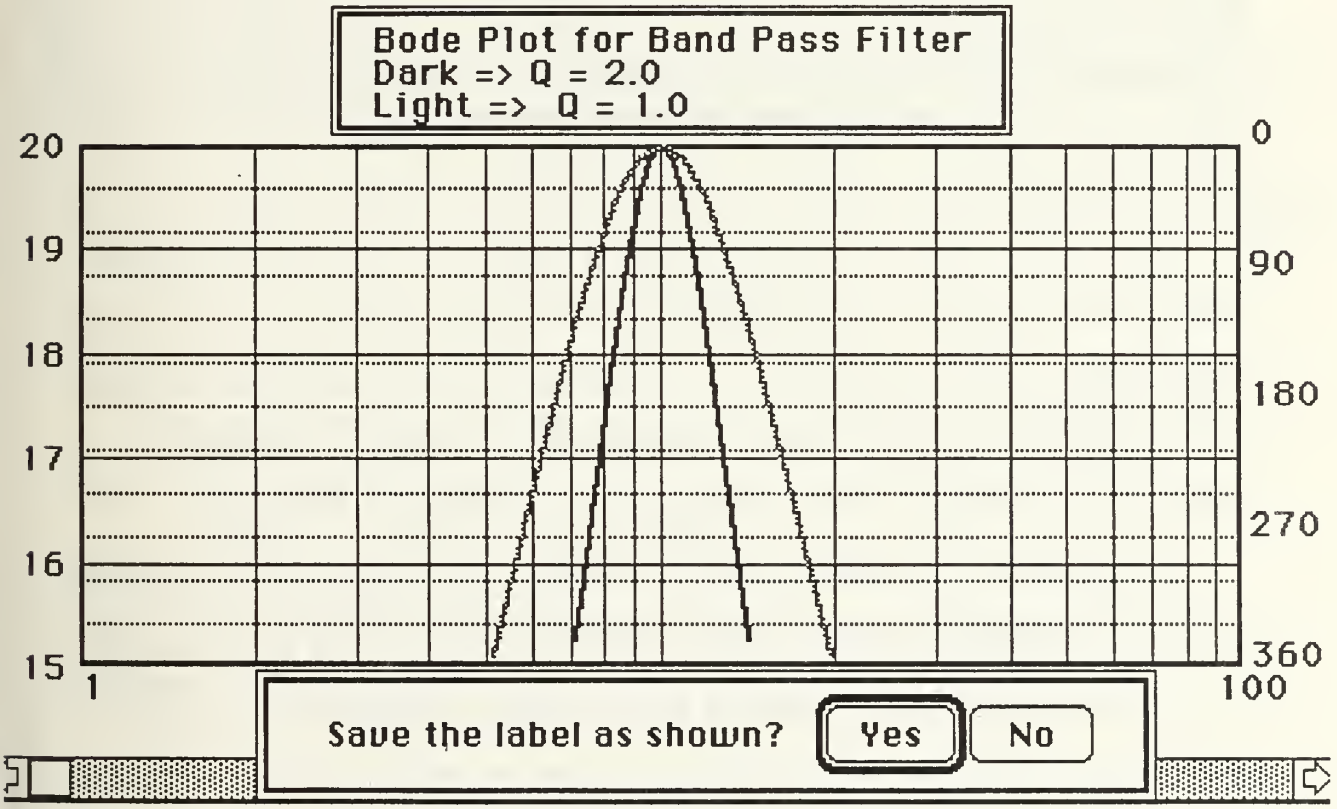
Fig(10) Overlapped Bode Plot With New Q = 1.0

A label on top should show that it is a Bode plot and identify which curve has which Q value. This is done by selecting 'Add Label' from the Tools menu while the bode plot is being displayed. Fig(11) shows the 'Add Label' dialog box after we entered the labels we want.After selecting 'OK' the Bode plot window is again displayed. MacCAD now waits for the user to push the mouse button and then draws a rectangle that is the size of the label about to be added. The rectangle moves across the screen as the mouse is moved. This allows the label to be placed where ever it is desired and fits. We will place this label at the top of the plot. After releasing the

button, the label is drawn at that location and the user is asked if he wants to save the label as shown. This is shown in Fig(12).

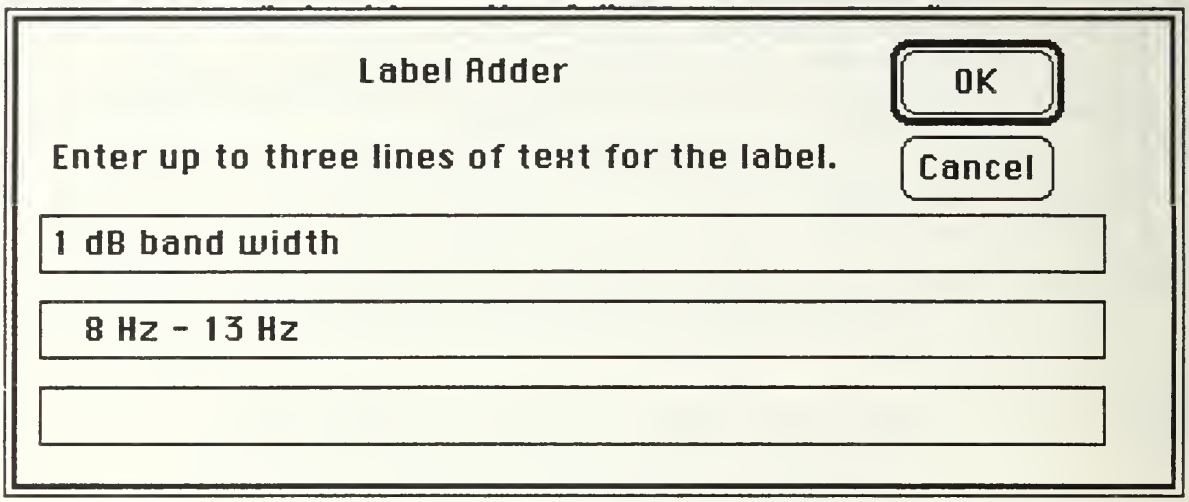


Fig(11) Add Label Dialog Box For Top Label.



Fig(12) Alert Box For Saving Label.

We will save the label by selecting 'Yes'. We can add another label showing the 1 dB band width corner frequencies. This is done in the same matter as the first label. The second label data is shown in Fig(13).



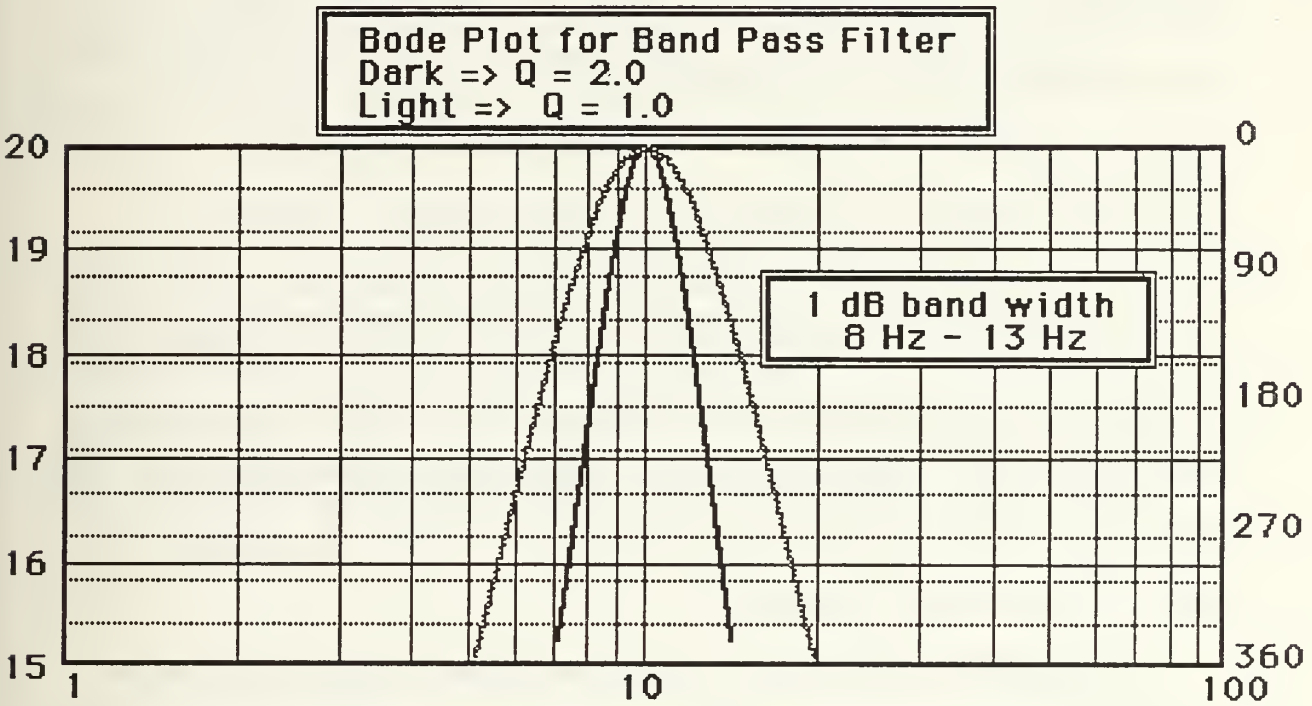
Fig(13) Label Data For Second Label.

Since the first line entered was the longer of the two, a couple spaces were added to the beginning of the second line so it would also be centered. This is of course, not necessary and if not done with the extra spaces would just show the second line start at the left position like the first. The final Bode plot is shown in Fig(14).

In conclusion, the Bode Plot tool allows the magnitude and phase curves to be plotted on one graph. The magnitude limits and frequency range of the plot can be selected by the user. Displaying the phase curve is also optional. An unlimited number of plots can be

superimposed, or overlapped as well as an unlimited number of labels added to the plot.

It must be remembered that any internal labels, within the plot rectangle may be plotted over if 'Overlap Plots' is selected after the labels are added.



Fig(14) Final Bode Plot.

To avoid this, wait till all the plots are displayed before adding any internal labels. Centering text in the labels was also demonstrated as being very easy. The 'Add Label' will work in the exact same way for all the other plots and graphs generated by MacCAD.

VI. NYQUIST PLOT

A. BASIC DESCRIPTION.

The Nyquist Polar Plot is very similar to the Bode Plot in that the phase and gain of G(s) are displayed as functions of frequency. The difference is that the Nyquist plot is in a polar coordinate system where the Bode plot is in rectangular coordinates. In the polar plot however, the response at any particular frequency is shown as a vector with it's magnitude equal to the gain of G(s) and an angle equal to the phase delay of G(s) at the same frequency. Unlike the Bode Plot which calculates gain in decibels, the Nyquist Plot shows gain simply as (output magnitude)/(input magnitude). The phase delay is shown as the angle measured from the positive 'x' axis, in a clockwise direction.

The system's stability can be measured by the gain and phase margins. Phase margin can be determined from the Nyquist Plot by noting where the plot crosses the unity magnitude circle. The angular difference between the crossing point and the -180 degree radial is the phase margin. This is automatically calculated for you by MacCAD. The point where the plot crosses the -180 degree radial shows the gain margin. Gain margin is defined as the reciprocal of the distance from the origin to the -180 degree radial crossing. This quantity is expressed in dBs as gain margin. This quantity is automatically calculated for you by MacCAD and will be explained later.

Another use for the Nyquist Plot however is to determine the number of closed loop transfer function poles in the right hand side of the 's' plane. This is done by determining the number of rotations the trace makes around the (-1, 0) point on the plot. The number of rotations, measured positive if in a clockwise direction, minus the number of open loop poles in the right hand plane gives the number of closed loop poles in the right hand plane. This is based on the Principle of Argument which Kuo [Ref 1] states as follows.

> Let $\Delta(s)$ be a single-valued rational function that is analytic in a given region in the s-plane except at a finite number of points. Suppose that an arbitrary closed path, Γ_s is chosen in the s-plane so that $\Delta(s)$ is analytic at every point on Γ_s; the corresponding $\Delta(s)$ locus mapped in the $\Delta(s)$-plane will encircle the origin as many times as the difference between the number of the zeros and the number of poles of $\Delta(s)$ that are encircled by the s-plane locus Γ_s.

If the closed path is chosen to cover the entire right hand s-plane, then the number of zeros minus the number of poles of the open loop transfer function that are in the right hand 's' plane will be indicated by the number of clockwise trace rotations around the origin. Since rotations around the origin are usually of infinitesimal radius, they cannot be seen on a graph drawn to scale. They can however be determined by viewing the Main System block data in factored form. A quick example will show how this information can be used.

Given an open loop transfer function, G(s), which has a zero order numerator and a second order denominator, the closed loop characteristic polynomial can be written as 1 + G(s) which we will call CP(s). Let Zo and Po be the number of open loop zeros and poles, respectively, in the right hand plane. Let Zc and Pc be the number of zeros and poles of CP(s) in the right hand plane. We know from inspection that Po = Pc since any value of 's' that makes G(s) go to infinity will also make 1+G(s) go to infinity. Let No be the number of rotations around the origin in the clockwise direction, and Nc be the rotations around the point (-1,0). If a Nyquist plot of our system showed two rotations around the origin in a counter clockwise direction, this would mean than No = -2. Since we knew there are no open loop zeros in the right hand plane because the numerator is zero order, then we would know that there must be 2 poles of G(s) in the right hand plane. This is shown in the equation:

$$No = Zo - Po = -2 = 0 - (2)$$

Let us say that there are also two counterclockwise rotations around the point (-1,0). This means that Nc also equals -2. Since Po = Pc, we know that Zo = 0 since;

$$Nc = Zc - Pc = -2 = 0 - (2)$$

By the definition of Zc, it means that there are no zeros of CP(s) in the right hand plane. Remembering that CP(s) is the characteristic

polynomial, and a zero for the characteristic polynomial is a pole for the transfer function, we then know that there are no closed loop poles in the right hand plane. Although this procedure seems cumbersome, it can be handy for determining how close to instability a system may be by counting the number of rotations around the point (-1,0). An illustrative example later in this section will show how this analysis tool can be used.

B. USER OPTIONS.

Data for the Nyquist Plot is calculated in much the same way as for the Bode Plot. Many of the same options are available. The user can select the maximum and minimum frequency for the calculations. Unlike the Bode Plot, these frequencies are entered as real numbers for the radial frequencies, rather than integers for exponential powers. Being a polar plot, the only plot dimension needed is the maximum radius. This is entered as any positive integer. The number of points to plot can also be set. The default value of 200 is usually sufficient for a smooth continuous curve but under certain conditions you may prefer more or less. Like the Root Locus, the Nyquist Plot also offers the option of Linear and Logarithmic intervals when selecting the value of frequency for each calculation. The advantages of either interval option is explained further in the section covering the Root Locus plot. As a rule of thumb, if you selected a wide frequency range, use the logarithmic interval. If the high end of the frequency range is of most interest,

use the Linear interval and if the low end is more important, use the Logarithmic interval.

As mentioned earlier, MacCAD calculates the phase margin and gain margin in the proper units for you. This information is displayed in the data box appearing in the lower right hand corner of the Nyquist Plot window. It also displays the input frequency and output phase angle for the output magnitude values of .5, 1.0, 2.0 and 3.0 The method of these calculations will be discussed in the Programmers' Notes of this section. It is assumed that the user can recognize a non minimum phase transfer function and he will realize that even though phase and gain margin data may still be displayed, by definition they do not exist for non minimum phase systems and cannot be used to determine stability in the same manner as with minimum phase systems.

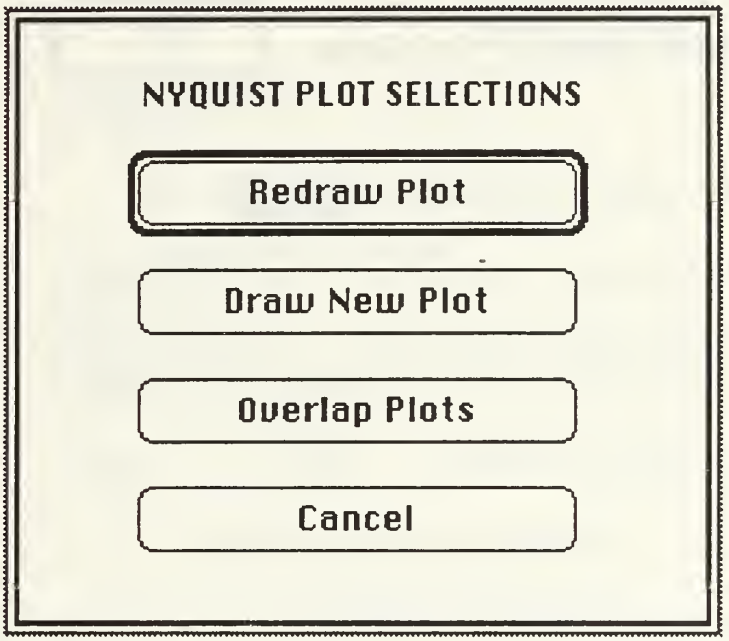
C. ILLUSTRATIVE EXAMPLE 1.

The following single block transfer function will describe our system for this example.

$$G(s) = \frac{(s + 2)}{(s^3 + 3s^2 + 10)}$$

Cascaded with this system is step adjustable amplifier with settings of 0, 4, 8, 12, 16 ... The Nyquist plot will be used to determine the lowest gain setting which will ensure closed loop system stability. G(s) will be entered and the numerator gain constant will be adjusted through the amplifier gains. This will

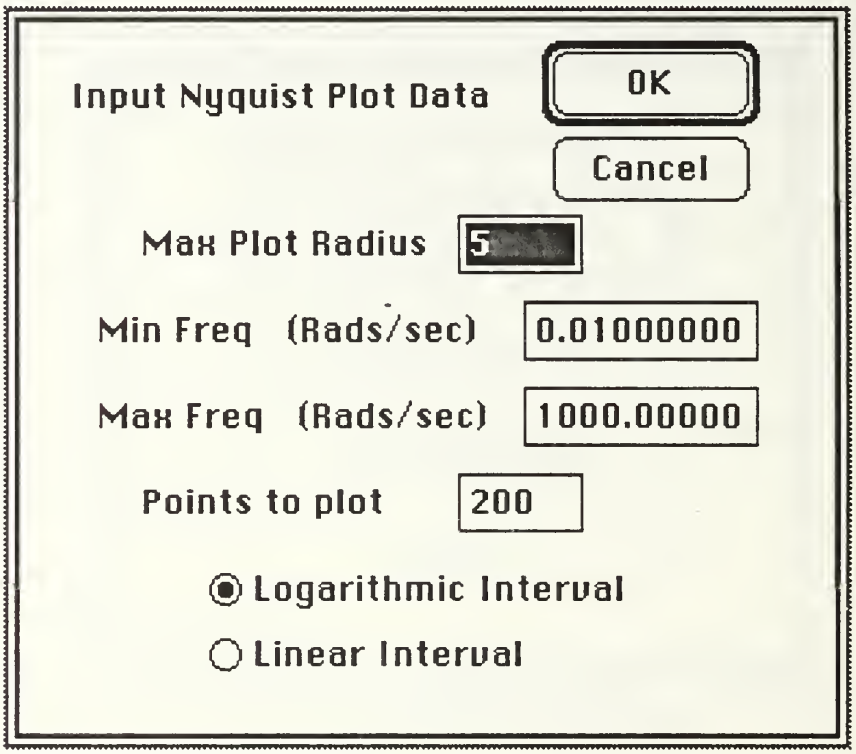
illustrate the use of the Nyquist plot and the procedure mentioned earlier in this section. We will first enter 4 for the amplifier gain and draw the Nyquist Plot. After G(s) has been entered, we select 'Nyquist Plot' from the 'Tools' menu. Fig(1) shows the Nyquist Plot dialog box.



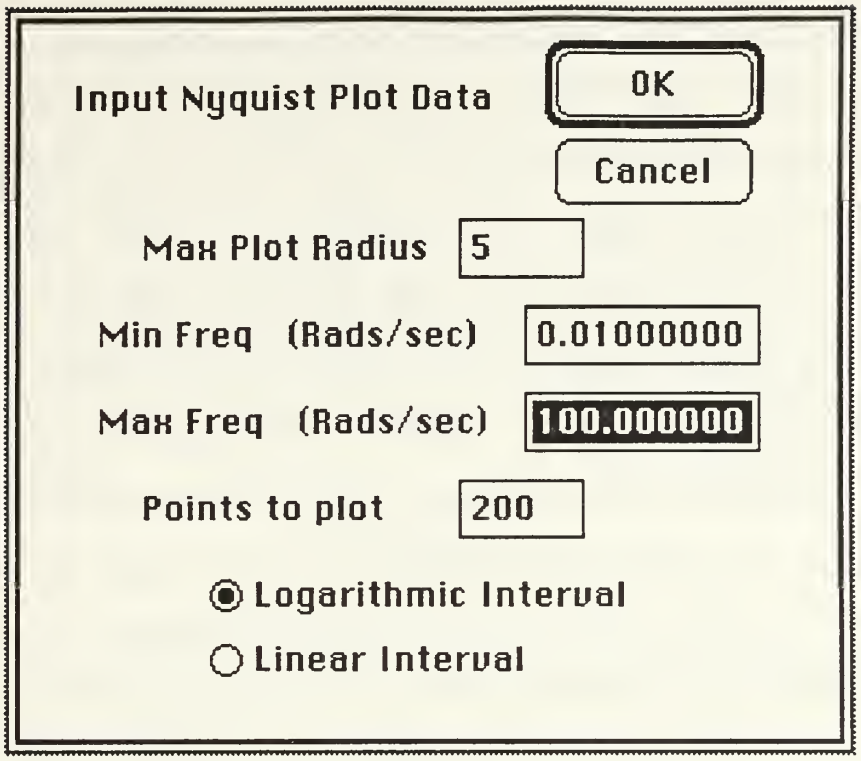
Fig(1) Nyquist Plot Dialog Box.

Select 'Draw New Plot' since this is the first plot. You will then see the dialog box of Fig(2). A smoother plot will be drawn if more points are plotted or if a smaller frequency span is selected. Plotting more points also increases the time required to do the plot calculations. Since all the numbers in the transfer function are between 1 and 10, it is a reasonable assumption that we do not need to go all the way up to a maximum frequency of 1000 radians per

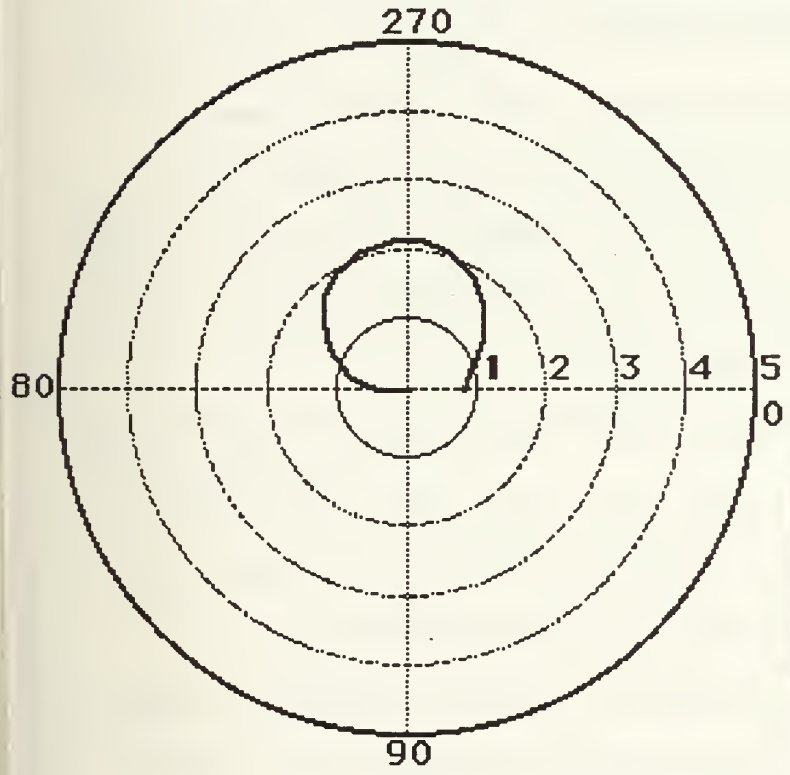
second. In this case we will use 100 for the max and keep all the other default settings. After entering our parameters, the dialog box appears as in Fig(3). After selecting OK, a countdown alert box shows the number of points to be calculated and then the plot is displayed as in Fig(4).



Fig(2) New Plot Dialog Box.



Fig(3) Parameters Of First Plot.



Nyquist Plot
Amp Gain = 4.0

Phase Margin (deg) = -15.58		
Mag	Phase	Freq
0.5	182.8	2.884
1.0	195.6	2.291
1.5	211.3	1.995
2.0	239.2	1.738

Fig(4) First Plot.

The direction of the rotation can be by examining the system transfer function. If the system is type zero, the trace will start from the 0 degree radial direction. A type one system will start from the -90 degree radial position, type two from the -180 degree radial and so on. The trace will usually end up at the origin if the max frequency is high enough. The direction that the trace approaches the origin from can also be determined from the transfer function. Subtracting the numerator order from the denominator order will give a number that identifies the origin approaching direction in the same way the system type identifies the starting direction. Knowing the starting point and the ending point of the trace will let you figure out the direction of rotation. Since the points are calculated from the minimum frequency to the maximum frequency, the rotation direction can also be determined by watching the trace as it is drawn. If you missed it the first time, select Nyquist Plot from the Tools menu again and select 'Redraw'. The complete Nyquist plot would also cover negative frequencies but the resulting plot is a mirror image of the positive frequency plot along the horizontal, 'x' axis so it is not drawn.

We also notice from the data box in the lower right corner, that when the magnitude is .5, the frequency is only 2.884. This means that we can decrease our max frequency parameter from 100 to 10. This will improve the accuracy and smoothness of the plot.

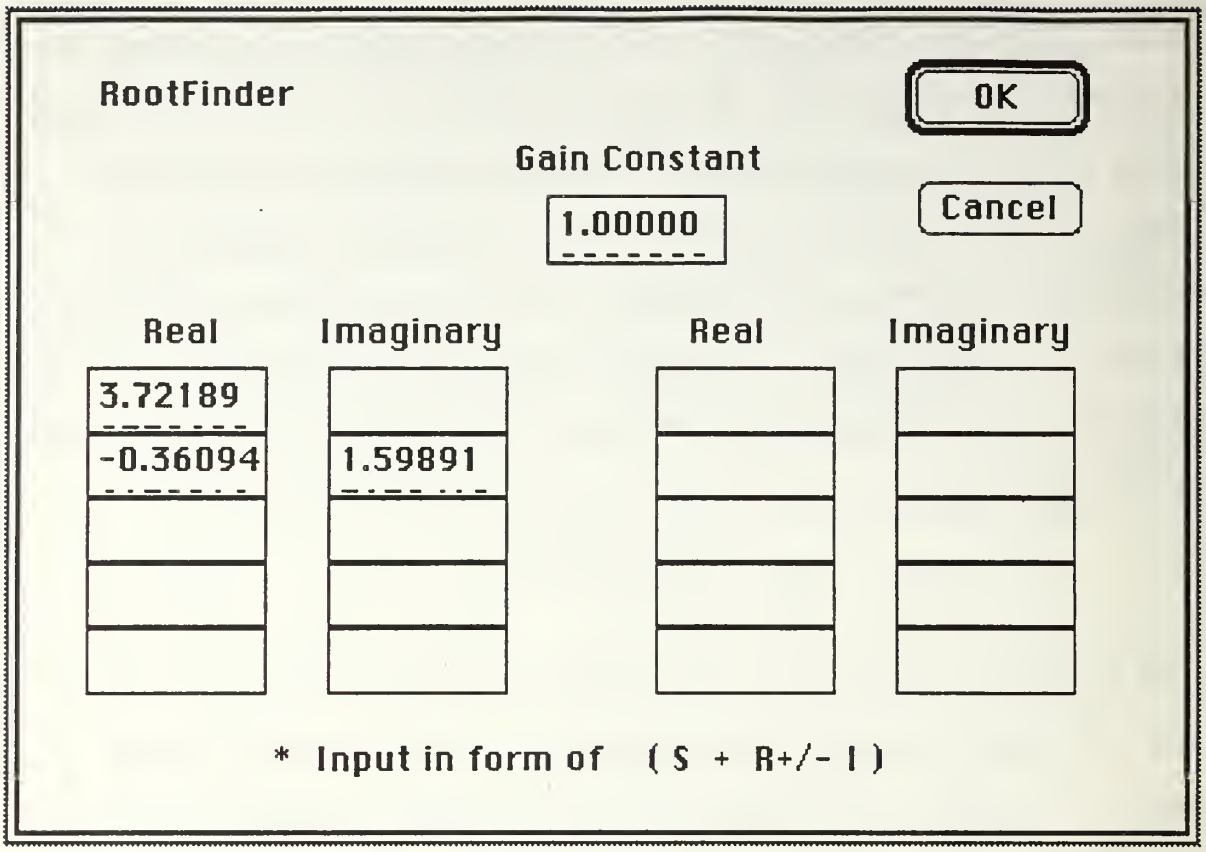
As mentioned earlier, the number of rotations around the origin cannot be seen from the plot but the same information can be seen

found from the original open loop transfer function. The first order numerator (s+2) obviously does not contribute any zeros in the right hand plane. The denominator (s^3+3s^2+10) can quickly be checked using the Rootfinder tool. Fig(5) shows the result of using the Rootfinder. It shows a complex pair in the right hand side. Since Zo = 0 and Po = 2 then No = -2. We can see from the Nyquist Plot in Fig(4) that the (-1,0) point is not circled at all. This means that Nc = 0 and since we know;

$$Po = Pc = 2$$

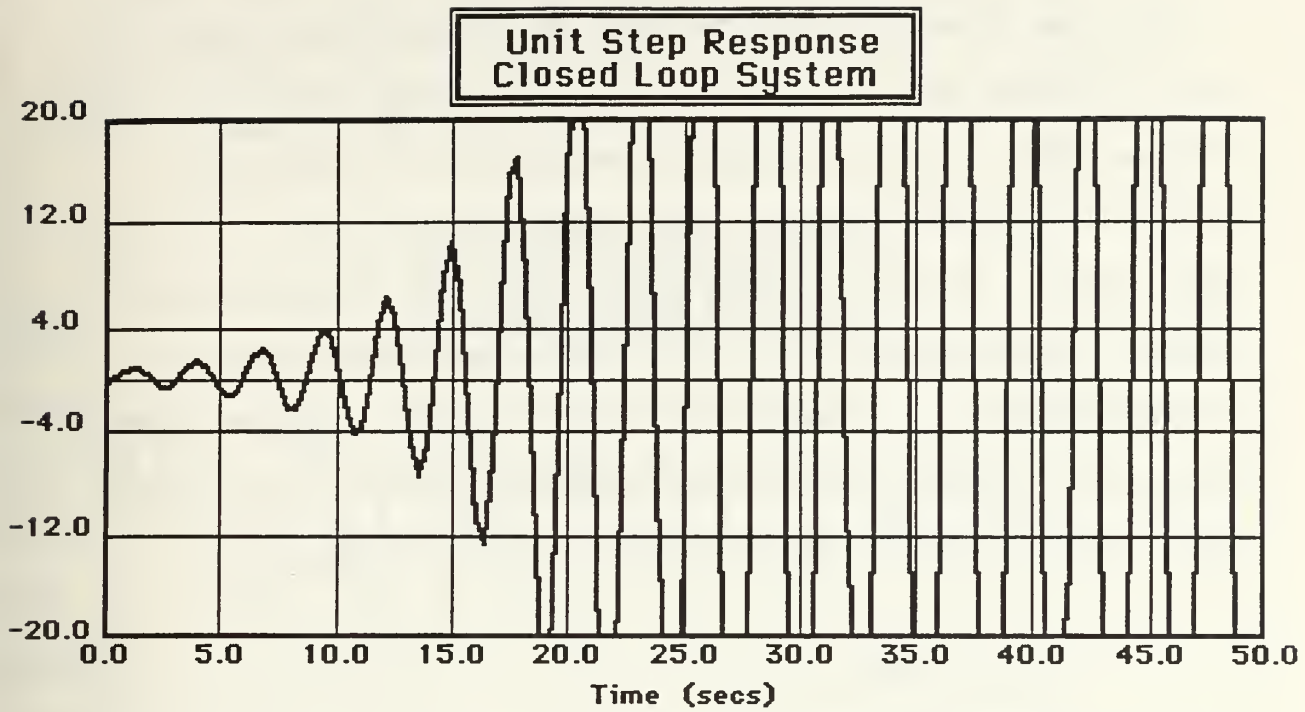
we then know that Zc = 2. This tells us that CP(s) has two zeros in the right hand plane so the closed loop transfer function has two poles in the right hand plane making it unstable. This can be checked by a variety of ways. The unit step response of the closed loop system is shown in Fig(6).

The closed loop system is clearly unstable with a gain of 4. Before changing the amp gain, the Nyquist plot can be drawn again using a maximum frequency of 10 instead of 100 as mentioned earlier. The plot does not appear very different so it is not reproduced here but we know that the data box information will be more accurate.

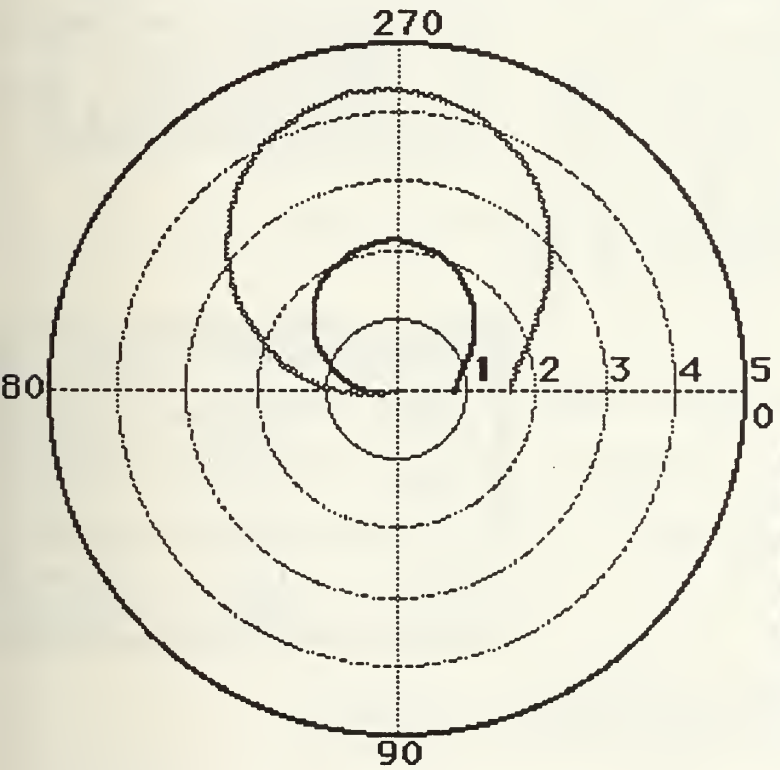


Fig(5) Factored Open Loop Transfer Function Denominator.

Using the 'Blocks' item, 'Change', the amp gain is then increased to 8. The Nyquist Plot Tool is again selected but this time 'Overlap Plots' is selected. The same parameters are used as with the last plot and the new one will be drawn on top of it. Fig(7) shows the overlapped plots. The plot appears to be close to rotating around the (-1,0) point but not quite. This can be checked by the data box which shows the phase at the unity magnitude point to be 182.8 which means it is not circled. It would have to be below 180 to indicate it had been circled.



Fig(6) Unit Step Response Of Closed Loop System With Gain = 4.0

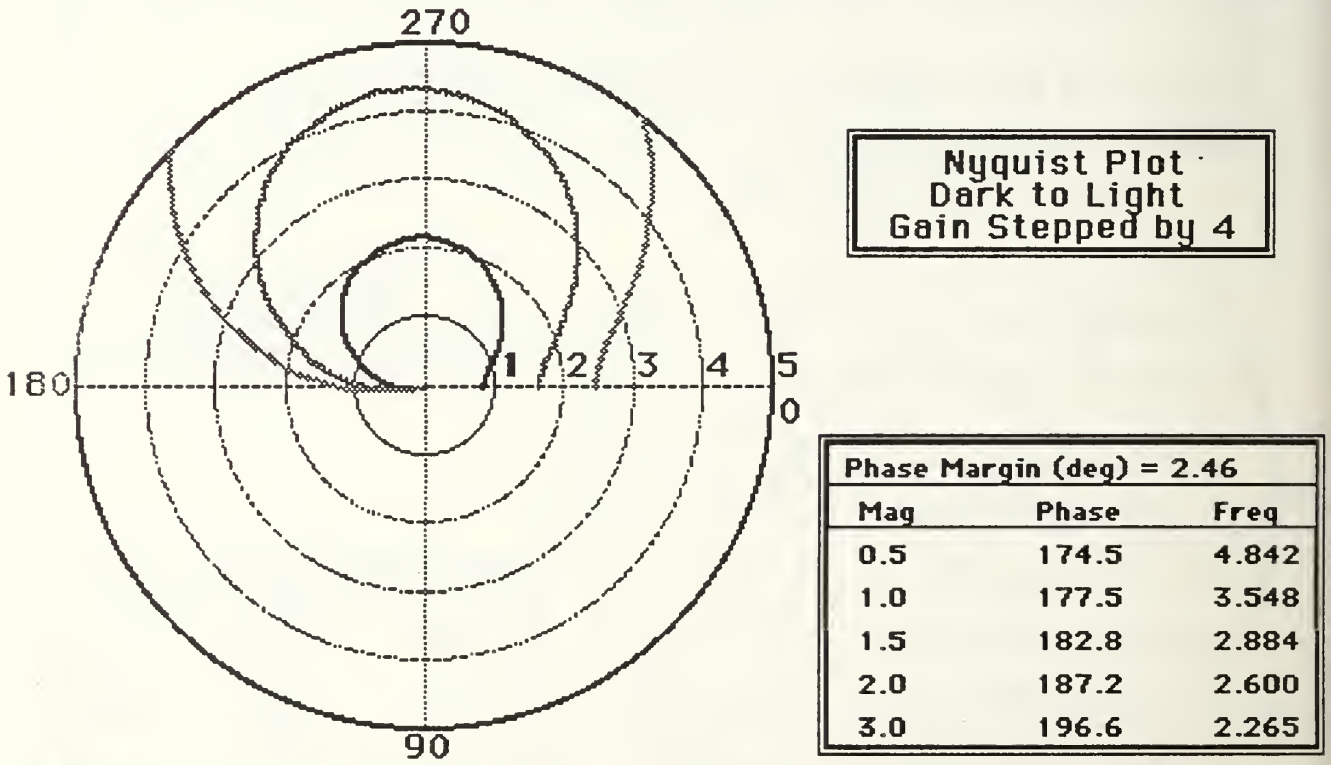


Nyquist Plot
Dark to Light
Gain Stepped by 4

Phase Margin (deg) = -2.80		
Mag	Phase	Freq
0.5	176.1	3.936
1.0	182.8	2.884
1.5	189.1	2.512
2.0	196.6	2.265
3.0	213.1	1.972

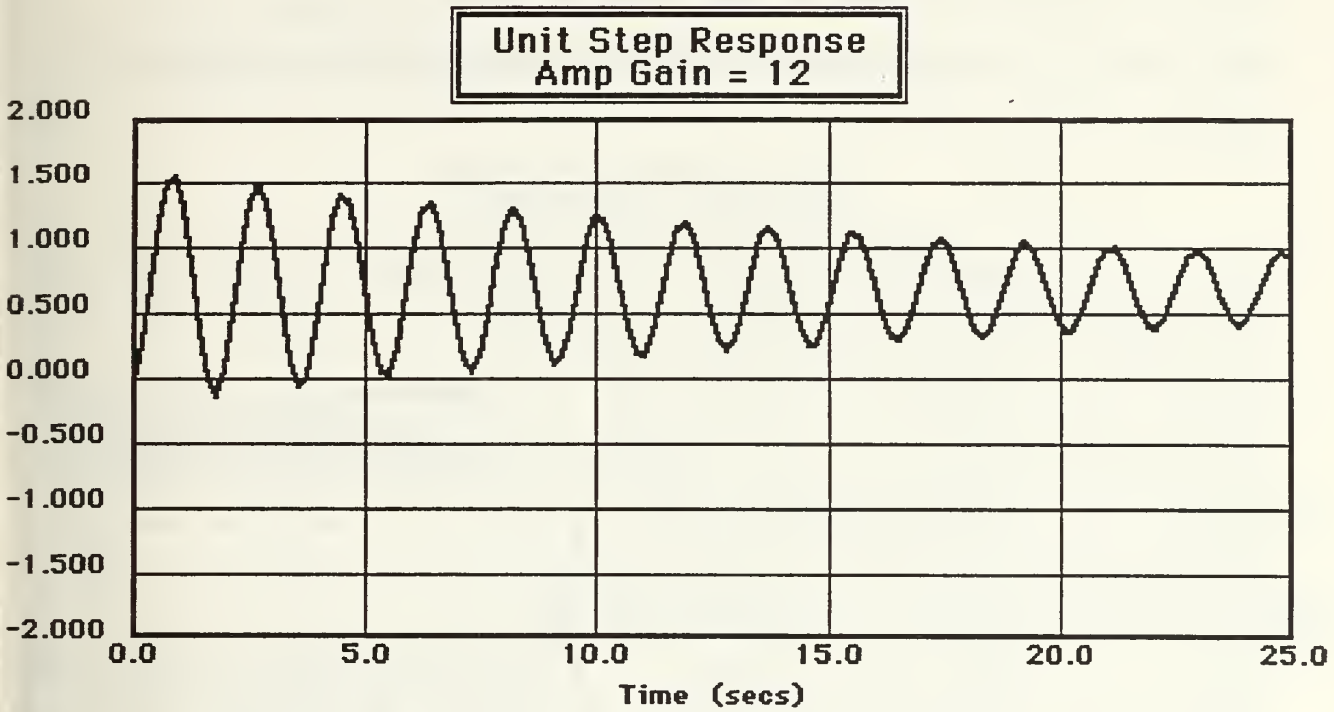
Fig(7) Overlapped Plots With Gain = 4 and 8.

We now know that although close, the closed loop system will still be unstable with the amp gain set at 8. Using the 'Blocks' item 'Change' again, the gain constant is increased to 12. Selecting 'Nyquist Plot' and the overlap option again gives the plot shown in Fig(8). Here it appears that the (-1,0) point was circled this time although it is still close. The data box confirms that it was encircled because the phase is 177.5. The stability is again checked using the Unit Step Time Response available under the 'Tools' menu. Fig(9) shows the response to be stable but slow in decaying.



Fig(8) Overlapped Plots With Gain = 4, 8 and 12.

This example has shown how to use the Nyquist Plot tool and has shown the use of the 'Overlap Plots' option and how to determine stability based on the Principle of Argument.



Fig(9) Unit Step Response With Amp Gain = 12.

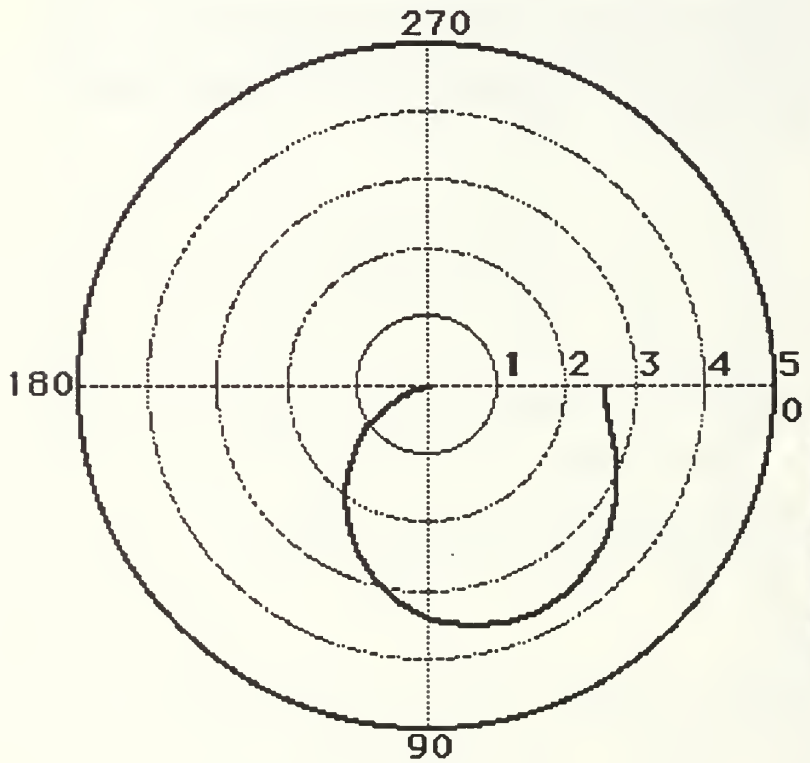
D. ILLUSTRATIVE EXAMPLE 2.

This will be a simple example using a minimum phase system and examining the data box information. The Nyquist Plot will be compared to the Bode Plot. The system has been entered and the transfer function is;

$$\frac{10}{s^2 + 1.5\,s + 4}$$

This is a second order underdamped system with $\zeta = .75$ and $\omega n = 2$. The Nyquist plot for this system is shown in Fig(10).

As mentioned in the previous example, the data box shows phase and frequency information for various magnitude values. It also shows phase and gain margin data. For a second order system, like this one, the plot never crossed the -180 degree radial so the gain margin is actually infinite. In this case, it is not displayed in the data box. As a comparison, the open loop bode plot will be shown in Fig(11).



2nd Order System

Phase Margin (deg) = 31.78		
Mag	Phase	Freq
0.5	160.4	5.012
1.0	148.2	3.548
1.5	141.7	3.162
2.0	127.7	2.661
3.0	106.8	2.239

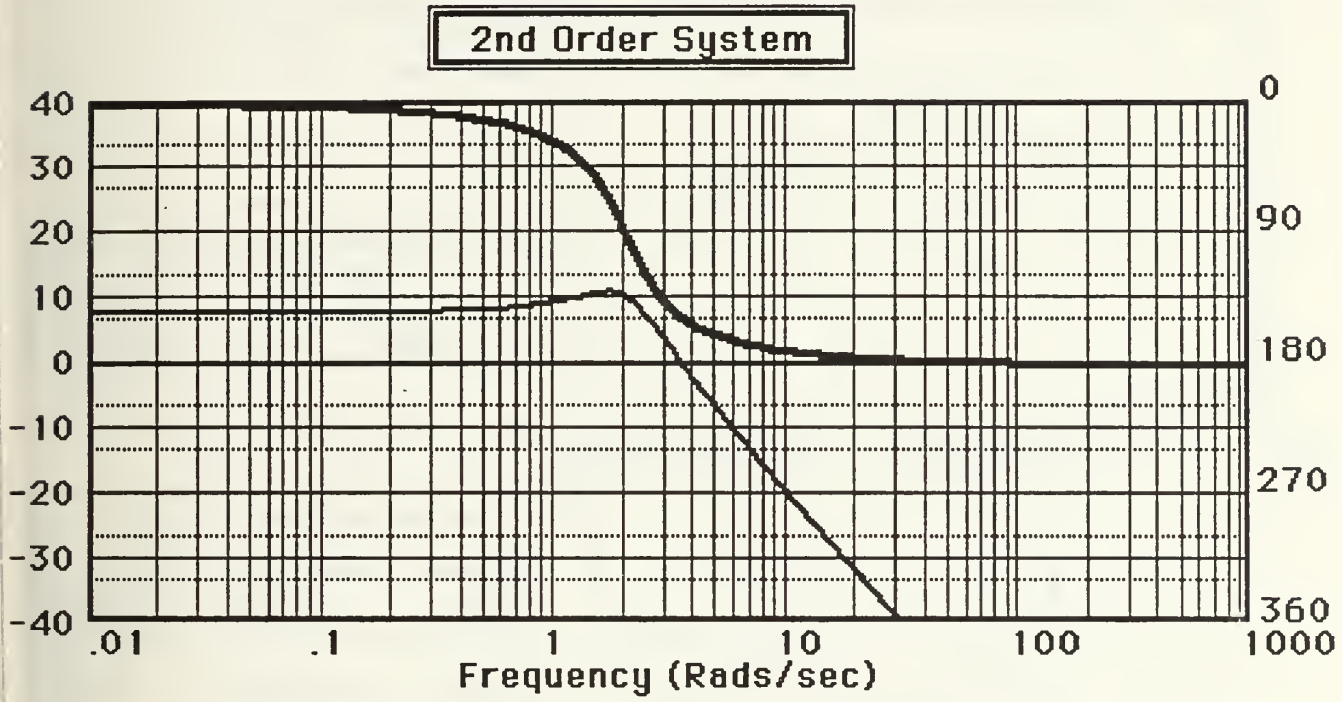
Fig(10) Nyquist Plot of Second Order Underdamped System.

As can be seen from the Bode plot, the system is stable with a gain margin of about 30 degrees. To illustrate the gain margin, a

pole at zero is added to the system. The transfer function is now;

$$\frac{10}{s(s^2 + 1.5\,s + 4)}$$

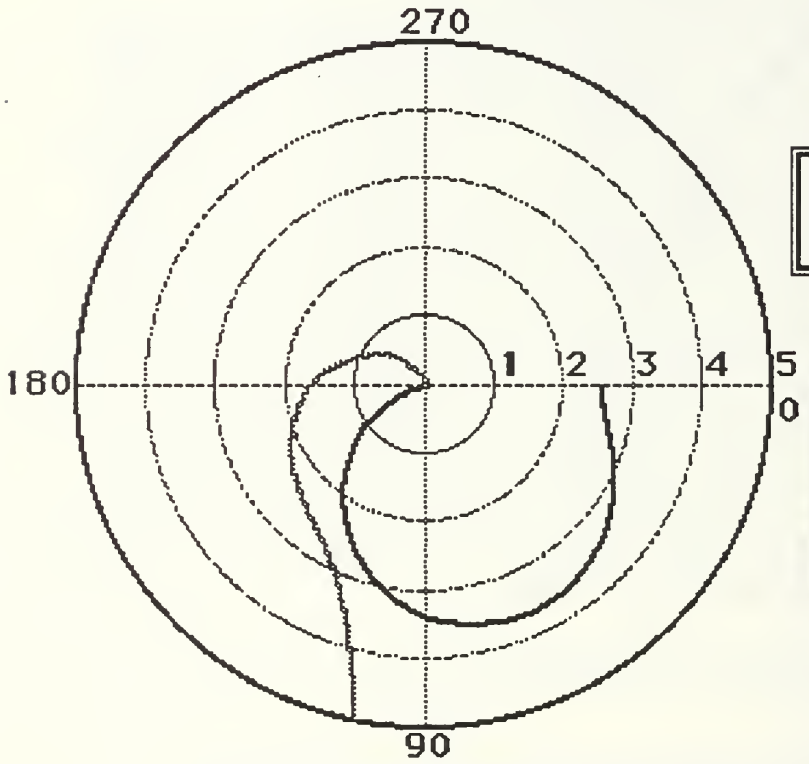
The overlapped Nyquist plot is shown in Fig(12).



Fig(11) Bode Plot For Second Order System.

As expected, the new type one system approaches from the -90 degree position and 'enters' the origin from the -270 degree position. In this case, the -180 degree radial is crossed when the gain is about 1.5. The data box shows the gain margin to be -3.39 dBs which correlates to the observed gain. Using the method discussed earlier in this section which determines the number of poles of the closed

loop in the right hand plane, we know from the open loop transfer function that Po = 0 since any quadratic with all positive coefficients will always have positive factors and the pole at zero is not considered in the right hand plane. Folding the third order Nyquist plot horizontally, to include the negative frequencies, we see that the (-1,0) point is rotated twice, clockwise. Since Pc = 0 and Nc = 2, this means that Zc = 2, which means the closed loop transfer function has two poles in the right hand plane.



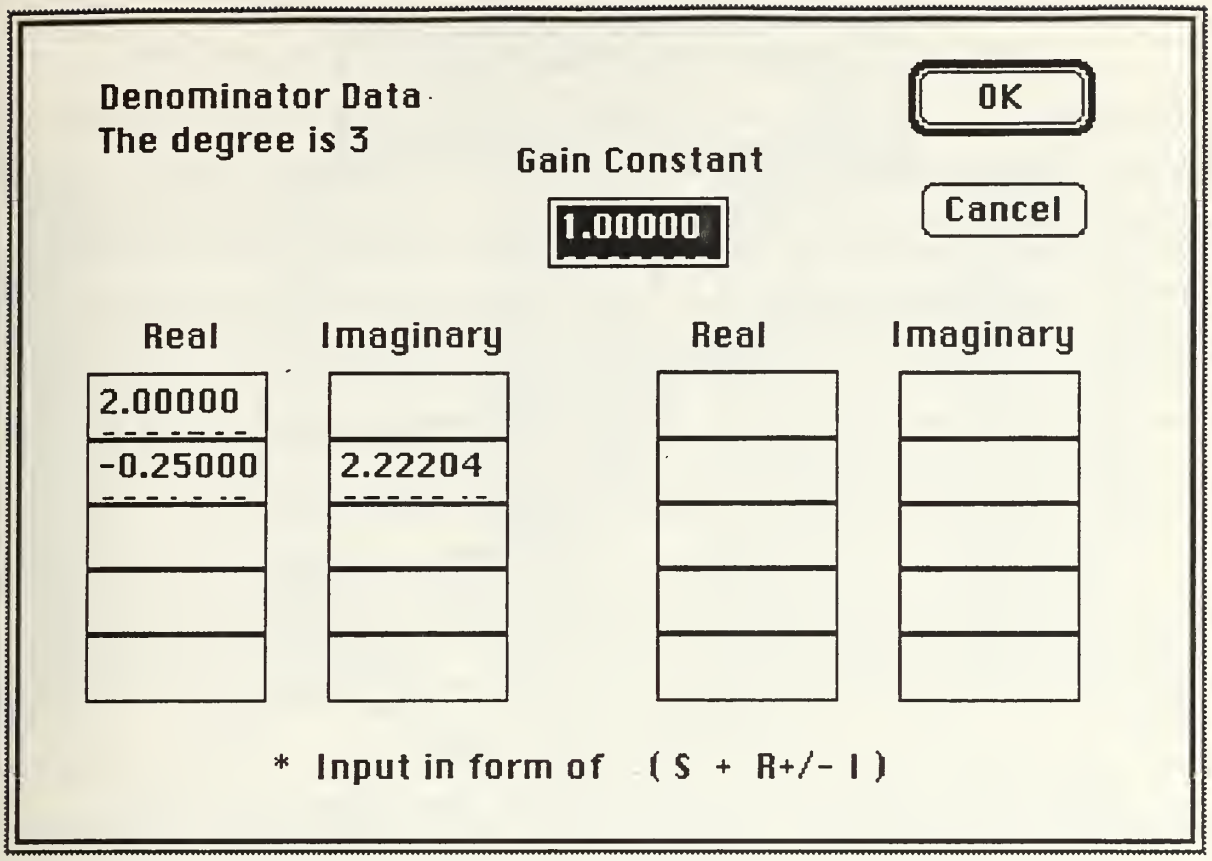
Nyquist Plot
Dark-2nd Order System
Light- (1/s) Added

Gain Margin (dB) = -3.39
Phase Margin (deg) = -31.51

Mag	Phase	Freq
0.5	231.7	3.162
1.0	211.5	2.512
1.5	188.4	2.113
2.0	170.9	1.884
3.0	116.6	1.000

Fig(12) Overlapped Second And Third Order Nyquist Plots.

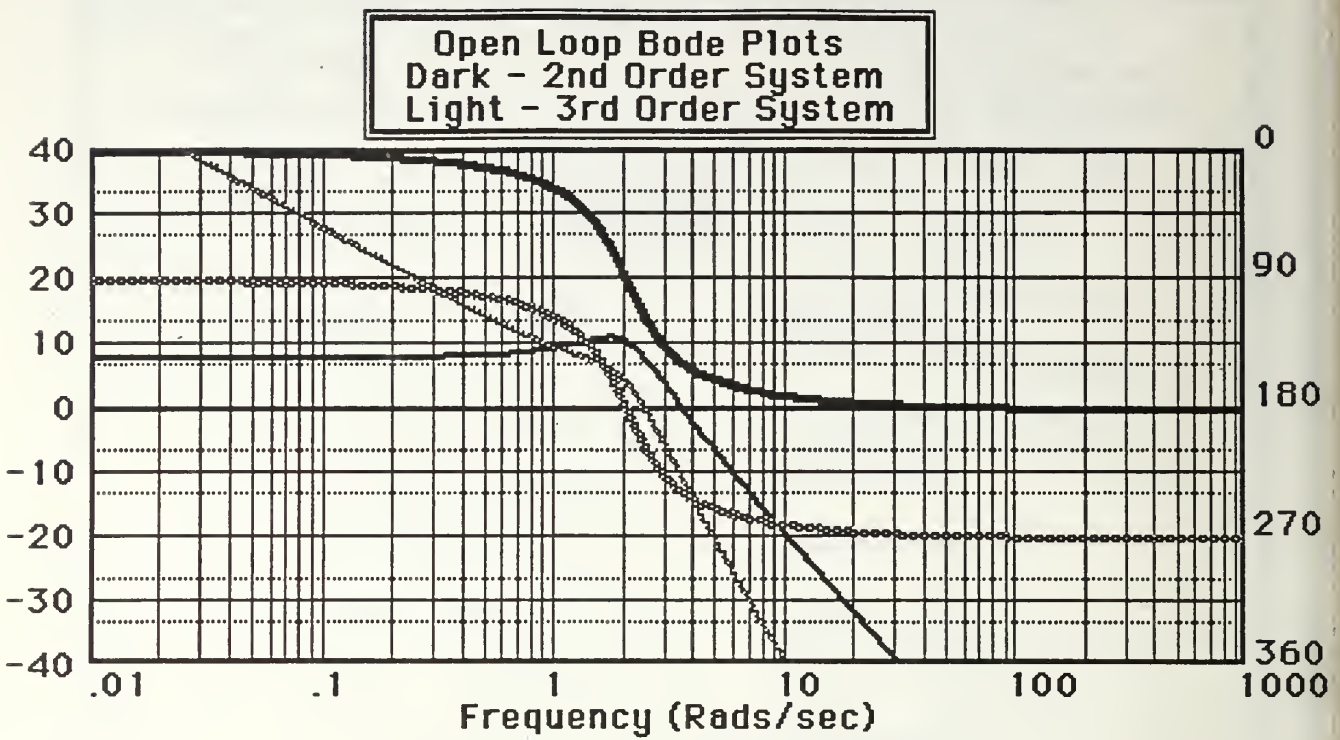
This is checked by examining the System Block in factored form using the 'Blocks' menu. First the system loop path is changed to 'Closed Loop' to add the unity feedback. The System Block data can then be examined. The denominator data is shown in Fig(13).



Fig(13) Closed Loop System Denominator In Factored Form.

As the Nyquist plot told us, the complex pair of factors have negative real parts indicating 2 poles in the right hand plane. The open loop Bode Plot is also overlapped in Fig(14) to compare the new system.

As can be seen from the light open loop Bode plot, the phase crosses below -180 degrees before the gain crosses below 0 dB. This is an easy check which tells us that both phase and gain margin are negative, so the system is unstable. This again is confirmed by the Nyquist plot which says the the gain margin is -3.39 and the phase margin is -31.51.



Fig(14) Overlapped 2nd And 3rd Order Open Loop Bode Plots.

E. PROGRAMMER'S NOTES.

As mentioned earlier, the magnitude and phase of the transfer function with various frequencies input, is calculated using the same procedures as the Bode Plot tool. The overall algorithm for the entire Nyquist plot procedure is also basically the same as described in the Programmers' Notes covering Root Locus. A unique operation during the calculation of the Nyquist data points is calculating the information to be displayed in the data box. A data type called pointdata is defined as;

```
pointdata = record
     donedata, wasabove, isabove : boolean;
     phasept, freqpt, magpt : extended;
end;
```

There are six variables that are of this type. They are phasemarg, gainmarg, maghalf, magthreehalf, magtwo and magthree. Each variable corresponds to a line of information displayed in the Nyquist data box and has a flag amount. For example, the flag amount for magthreehalf is a magnitude value of 1.5. As each point to plot is calculated for the Nyquist plot, the flag for each of the 6 variables is checked. The boolean fields wasabove and isabove are set and cleared as the flag variable (magnitude for magthreehalf) hits the flag amount. When the flag is hit, the magnitude, frequency and phase corresponding to the flag amount are saved in the fields phaset, freqpt and magpt. The flag amount only triggers the loading of the data if it is hit while decreasing. This means that if the magnitude is increasing and passes through 1.5, it will not load the data for magthreehalf. This is done only when the magnitude is decreasing through 1.5. The phasemarg flag is magnitude = 1.0 and the gainmarg flag is phase = 180. Phasemarg data is also used for the 1.0 magnitude data line in the data box.

The Nyquist plot curve is actually made up of several straight lines connecting points calculated from 'EvalGeq'. The function 'PointInPlot' checks to see if both points will fall within the plot radius. If both do not, the line is not drawn. This is to prevent the possibility of a stray line crossing the plot. This could happen if too few points are plotted or there are large changes in phase or magnitude between adjacent points.

As mentioned in the Root Locus section, 'Cliprect' is used to prevent drawing lines outside of the desired plot area. With the Nyquist plot however, this is not a rectangle so the procedure 'SetClip' is used. The 'InitPlotStuff' procedure defines 'plotclipH' which is a handle to the region defined by the outer circle of the Nyquist plot. This was done by a procedure called 'FrameCircle' that is in the 'NumberCrunch' module. For some reason, it is not included in the Apple QuickDraw library. 'InitPlotStuff' is called only the first time the 'Nyquist Plot' tool is called. It also calculates the radius of the plot circle and the coordinates of the circles center. Although these calculations could have been replaces by constant values, it was left this way so if in the future, the plot dimensions or layout needed changing, it could be easily done in this one procedure rather than having to adjust all graphic calls.

'FrameCircle' is also used to draw the concentric circles making the radial grid of the plot. This is done in the 'DoRadialGrid' procedure which also calls from the 'NumberCrunch' module, the function 'FindSep' which will be further discussed in a later chapter but in short, it returns the optimal step size for 'tics' between some input maximum and minimum values. This is used to return integer radial values of the concentric circles used for the Nyquist plot grid. If a max radius of 20 was input, 'FindSep' will return a step size of 5, so there will be 4 circles, each with integer magnitude values. If a value of 10 is input, 'FindSep' will return a step size of 2 so there will be 5 circles.

The 'DoDataBox' procedure draws the data box in the lower corner. It determines how many lines will be needed and the size of the box based on how many points were calculated. Before actually drawing the box or it's data, the procedure 'EraseRect' is called which erases any previous data boxes drawn in the case of overlapping plots. 'WriteFreq' is a procedure that determines the size of the frequency value to be drawn and then moves the pen position accordingly before calling 'WriteDraw'.

Phase and magnitude data is calculated using the function, 'EvalGeq', just like for the Bode plot. The returned magnitude is multiplied by 'multfactor' to scale it. It is then entered in the procedure 'Pole2Rect' to determine the x and y coordinates for the screen. A line is drawn from the new point to the last point, forming the curve on the plot.

F. USERS' TIPS.

It must be remembered that regardless of the plot radius parameter input, graphic calculations are being done over the entire frequency range selected. This means that even though some data does not appear on the screen, plot points are still being used. The more points that are used that actually appear on the plot, the smoother the plot will be. If the plot contains any straight lines rather than all curves, it is a good indication that the density of points in the plot area is not great enough. There are ways to increase the density.

There is a slim chance that at one end of the frequency range selected, at either the first or last point plotted, there may be large changes in phase or magnitude between that point and the one next to it. If this occurs, you may see a straight line starting inside the plot and extending to the plot edge without any curves at all. This occurs because MacCAD will only draw a line if one of the two points defining that line lie within the radius of the plot. This single line, or any other perfectly straight lines appearing on the plot, could be an indication of insufficient point density.

Even though the default frequency range will usually cover the area of interest, it may also cover too much area that is not of interest. By looking at the magnitudes of the numbers in the transfer function, you can often determine if you really need to go all the way from .01 to 1000 rads/sec. If not, decrease the max frequency or increase the min frequency so there will be more points displayed on the plot.

An option that may also improve the plot is selecting 'Linear' or 'Logarithmic' intervals. The methods of calculating the interval is discussed in the Root Locus chapter but as a rule of thumb, if the ratio of the maximum frequency to the minimum frequency is greater than about 100, the Logarithmic option may be better. This is why it is the default setting. If you are more interested in the upper frequency area, such as a resonant peak, the Linear option would be better and likewise, if the low end of the frequency range is most important, the Logarithmic interval would be best.

The default is set for 200 points to plot. This will give good results in most cases and will only take about 30 seconds to plot. If adjusting the frequency range or the plot interval still does not give a smooth looking plot, try increasing the number of points plotted. Realize that doubling the number of points also doubles the time required to draw the plot.

If the trace rotation is counter clockwise, it is a good indication that the system might be a non-minimum phase system. This should be double checked using the methods discussed earlier on determining the direction of rotation along with examining the open loop transfer function.

When selecting to draw a new or overlapped Nyquist plot, it will be drawn using the Main System and it's loop path[1]. If you have just checked the closed loop Bode plot or time response, be sure to change the loop path back to Geq before calculating the Nyquist plot. Otherwise the plot will be confusing or meaningless. This applies to the use of all plots. You should always know what kind of loop path you have entered for your system before making a new plot.

As with any other plot in MacCAD, when using label boxes, if you think you may want to overlap plots and may want to change the label, make your first label with short lines of text and only 1 or 2 lines long. This way, after overlapping the plots, you can easily draw a new label over the old one by using more lines with longer text. It

---

[1] The type of feedback, Geq, Forward Path, Open Loop or Closed Loop. See chapter on Block manipulation for clarification.

may be difficult to draw a new label over an old one if they contain the same number of lines.

When analyzing higher ordered systems, the Nyquist plot may cross the -180 degree radial or the .5, 1, 1.5, 2 and 3 magnitude circles more than once. The data box may display surprising information. Remember how the data for each point is calculated. If you are not sure, reread the Programmers' Notes section in this chapter. MacCAD only reports the first time the requirements are met to calculate the data for each line entry in the data box. For example, some plots may cross the -180 degree radial more than once. If this is the case, the data shown in the box will be for the first time the plot crossed the radial in a clockwise direction. If it only crosses once in a counter clockwise direction, such as in the non minimum phase system used in the first example, the gain margin will not be displayed at all.

VII. ROOT FINDER

A. BASIC DESCRIPTION.

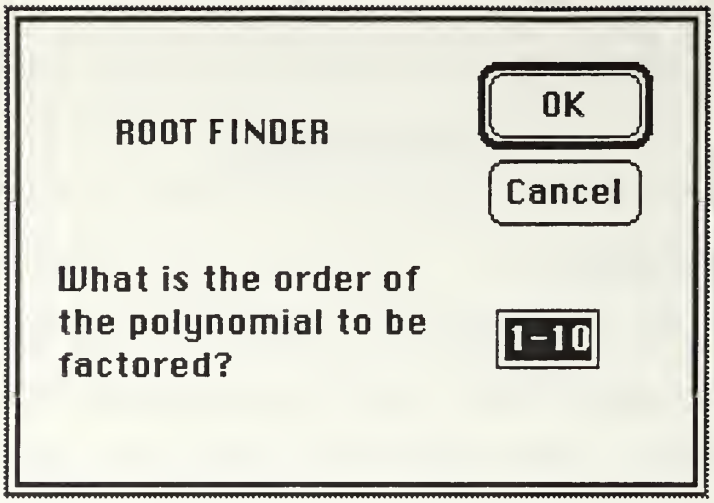
There are three cases where the roots of a large order polynomial, in coefficient form, must be determined. The first is when transforming a polynomial in the 'polycoef' variable type to the 'polyfact' type. This is because MacCAD saves all polynomial in the coefficient form but allows the user to enter or view them in either form. The second case is when calculating data points for the Root Locus plot. The characteristic polynomial, which is still in coefficient form, is solved using various system gain values. The third case is in the form of a tool available to the user at any time through the pull down menu item 'Root Finder' under the 'Tools' menu.

B. A SIMPLE EXAMPLE.

Consider having the task of finding the roots of the equation;

$$(s^4 + 2s^3 + 3s^2 + 4s + 5).$$

This will be done by using the 'Root Finder' function under the 'Tools' menu. The function is selected by either hitting the command key and the letter 'F' simultaneously or by selecting 'Root Finder' under the 'Tools' pull down menu. When this is done, you will see a dialog box like Fig(1), asking for the order of the polynomial that you would like to solve the roots for.

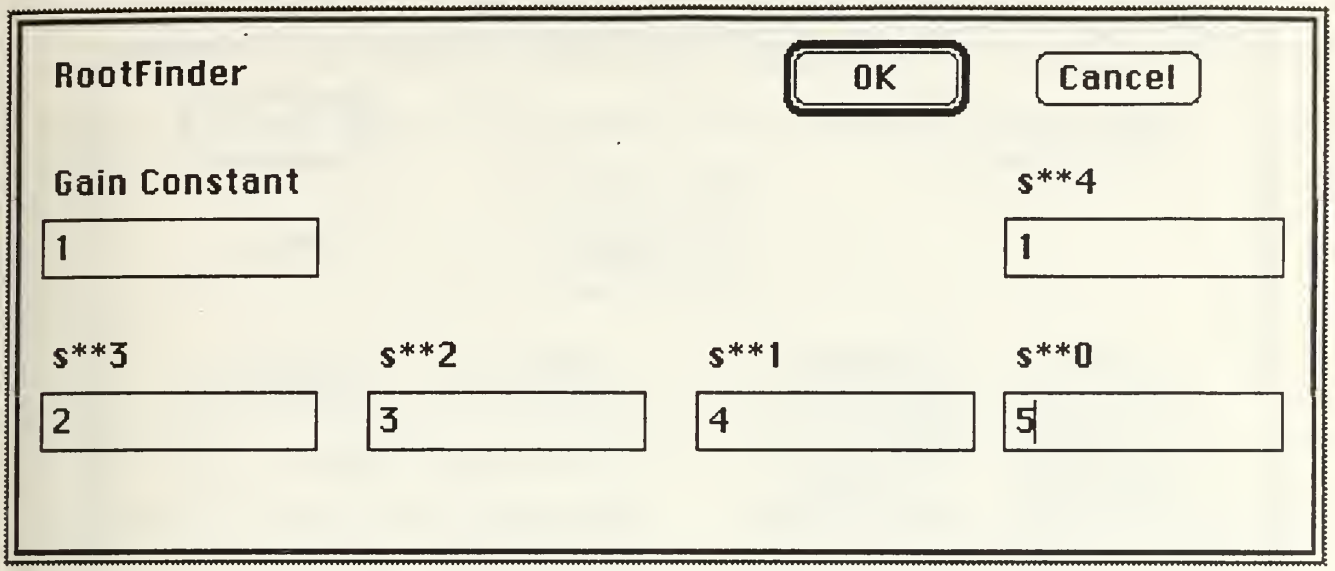


Fig(1) Root Finder Dialog Box.

Root Finder will presently calculate the roots for polynomials of degrees between 1 and 10. Entering any integer between 1 and 10 will start the 'RootFinder'[1] subroutine which will be described later. Any other characters will be considered an error. The data box will be outlined, the erroneous input highlighted and and the dialog box will wait for another input. In this case the integer '4' is entered, since the denominator is 4th order, and 'OK' is selected. A dialog box then appears asking for the coefficients of the polynomial. The coefficients of the polynomial are entered the same way as entering or editing block numerators or denominators, described in an earlier chapter. Fig(2) shows the dialog box after the data has been entered.

---

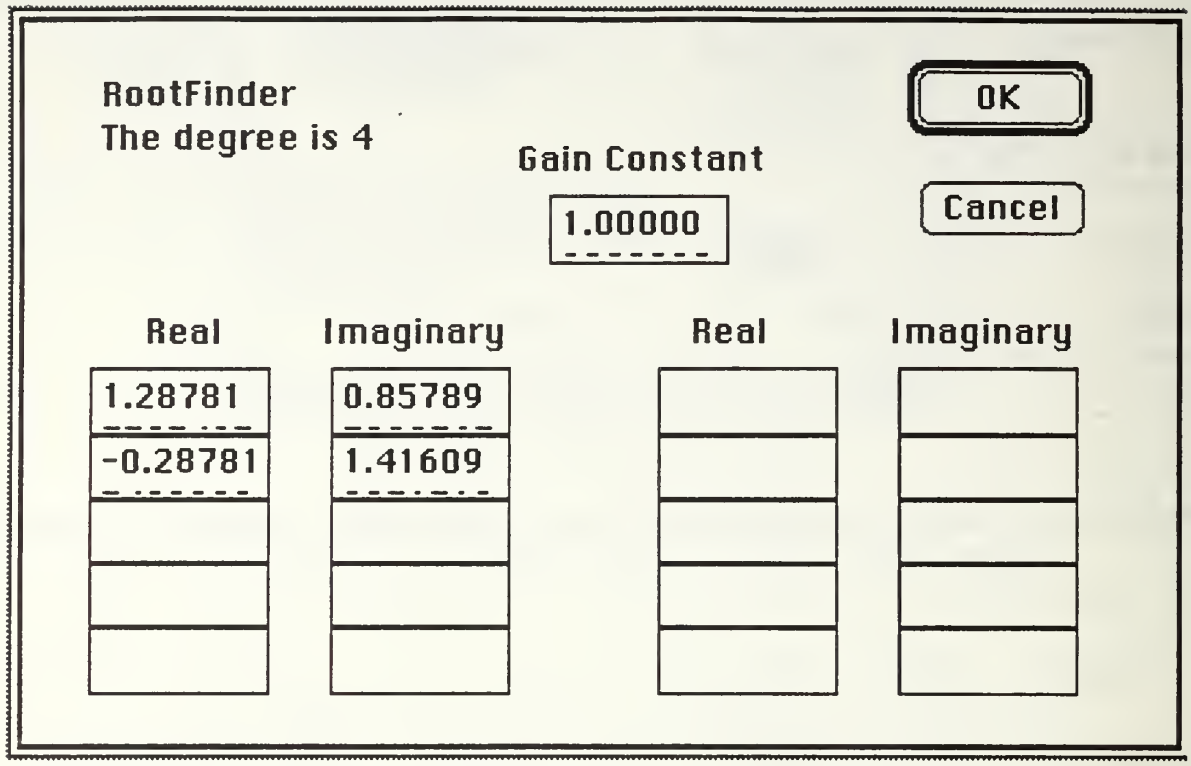
[1] The subroutine procedure is called "RootFinder" where as the tool available to the user is called "Root Finder". Note the use of spaces in differentiating the two names.



Fig(2) Third Order Equation Entered In Root Finder.

Although a gain constant will not affect the roots of the polynomial, it is still included in the dialog box in order to keep the format of entering polynomials constant throughout MacCAD. Any number can be entered in this case so the number 1 is used. The polynomial coefficients are entered in descending order and 'OK' is selected. The roots are then displayed in the same format as block data. Fig(3) shows the roots of the denominator.

The factors of the polynomial are shown to be two complex pairs. Remember that the roots of the equation will have the oposite sign of the factors displayed in the dialog box. If the third order equation was the denominator of an open loop transfer function, the fact that the real part of the second complex pair is negative indicates a pair of roots in the right hand 's' plane so the system would be unstable.



Fig(3) Roots Of G(s) Denominator.

C. PROGRAMMER'S NOTES: ROOTFINDER PROCEDURE.

The procedure which solves for the roots of a polynomial is called 'RootFinder'. It is is defined as;

```
RootFinder (polycin : polycoef;
    var   polyfout : polyfact
          accurate : boolean);
```

where 'polycin' is the polynomial in the coefficient form that is to be factored. 'polyfout' is the output polynomial in factored form. 'accurate' is a boolean input which when true, increases the accuracy requirements of 'RootFinder'.

'RootFinder' uses Bairstow's Algorithm which takes a high order coefficient polynomial and iteratively searches for a quadratic that can be divided into it. The quadratic that is searched for is in the form;

$$s^2 + Ps + Q$$

The values 'P' and 'Q' are set at an initial value, in this case zero, and the quadratic is compared to the original equation. Correction factors, delP and delQ, are calculated and added to the old values of 'P' and 'Q'. The quadratic is again compared to the original polynomial and corrections calculated and added. This process is repeated until the magnitude of the ratio of delP/P + delQ/Q is less that an acceptable error value, epsilon. When the accuracy requirements are met, the 'inlimits' boolean flag is set and the quadratic is solved with the quadratic equation through the procedure 'DoQuad' and the original coefficient polynomial which is recalculated has it's order decreased by 2. This entire process is repeated until the recalculated coefficient polynomial is of order 2 or less where the remaining roots are determined by 'DoQuad' if the order is 2, or by the remaining coefficient if the order is 1.

The value for epsilon is set at one of two values depending upon the accuracy desired. Higher accuracy requires more iterations which require more time. Calculating the roots for the Root Finder tool and changing a polynomial from 'polycoef' to 'polyfact' uses the higher accuracy epsilon, 10^{-7} which usually gives at least 5 digits of accuracy. The Root Locus plot calculations use the lower accuracy

epsilon, 10^{-2}, which gives about 2 digits accuracy. For plotting purposes, additional accuracy is not necessary since the difference could not be seen. Since fewer iterations are necessary for the lower accuracy, the plotting speed is greatly increased.

MacCAD's 'RootFinder' procedure was written starting with a previous PASCAL version of Bairstow's Algorithm written for the IBM personal computer by Wood. [Ref.2] Several changes were made to adapt to the Macintosh system and to use the data structures of MacCAD. In addition, three errors in the original code were also corrected.

The first was in the case of repeated roots. If there were more than two real roots at the same location, the 'P' and 'Q' values would not converge. They would oscillate around the values they should have converged to. The accuracy criteria is never met so the program is effectively in an infinite loop and only stops when the number of iterations reach a predefined maximum value. At this point the procedure would stop without updating 'polyfout'. The factor values contained in 'polyfout' would then be what ever number might have been residing in that memory location. Usually zero would be output or a floating point error would occur.

This error was corrected by decreasing 'delP' and 'delQ' by a factor of .5 any time both changed in sign from the previous correction. By decreasing the size of the corrections, 'P' and 'Q' were able to converge to the correct value. This was checked by inputting 3rd to 9th order coefficient polynomials with a variety of repeated

roots. Even with the highest accuracy setting, every polynomial was solved.

The second error occurred with roots at zero. It would appear at first that there would be little reason to want to solve a polynomial with roots at zero since they would be very obvious from the zero valued lowest coefficients. In the case of calculating data points for the Root Locus plot however, it is possible that for some value of gain, the characteristic equation could have roots at zero. In addition to this possibility, it was desired to make the procedure fool proof so the user could enter any polynomial, even one with obvious solutions, and the proper roots would still be found.

The error was easily corrected by checking the original coefficient polynomial to see if the lowest coefficient was equal to zero. If it was, the first factor value of 'polyfout' was set to zero and the original polynomial was 'shifted to the right' which eliminated the root at zero and decreased the order by one. This was repeated until the lowest coefficient was not zero.

The third error would not cause catastrophic results as would the first two but would give inconsistent accuracy. The original routine checked for accuracy by summing the magnitudes of 'delP' and 'delQ' and comparing that value to epsilon. If the actual values of 'P' and 'Q' were fairly small, the error criterion would be met with fewer iterations than if 'P' and 'Q' were much larger. In short, the error criteria was determined from the magnitude of the correction factors, 'delP' and 'delQ', regardless of the values of 'P' and 'Q'.

This was corrected by changing the 'inlimits' flag to be set when the relative changes of 'P' and 'Q' are less than epsilon. This was done by comparing the magnitudes of (delP/P) and (delQ/Q) to epsilon. In this way, the relative accuracies of 'P' and 'Q' are consistent before calling 'DoQuad'.

In summary, the 'RootFinder' routine solves the roots of any coefficient polynomial of order between 1 and 10. It is used by the MacCAD for converting polynomial types and when calculating Root Locus points. It is also available to the user in the tool 'Root Finder'. This allows the MacCAD user to pre-analyze data by being able to factor a polynomial of his choice before entering it as a block in the system.

VIII. ROOT LOCUS

A. BASIC DESCRIPTION.

The Root Locus is a widely used tool for determining the stability and response characteristics of a closed loop transfer function based on varying the gain of the forward path. The zeros of the transfer function are the roots of the characteristic equation. These roots are plotted with real component as the abscissa and imaginary component as the ordinate. The roots are plotted for each interval value of the gain from some determined minimum to a maximum gain value. The points form a locus and indicate how the roots move as the gain is increased. A variety of information is obtained from the plot. At any particular gain value, if any points are plotted in the right hand plane, the system would be unstable if that gain value were used. If all the roots lie on the real axis, then the system would not be expected to have any oscillations. If the locus of points is close to the imaginary axis, it indicates that the system is close to instability and a small change in any of the elements that make up the transfer function could lead to instability.

B. USER OPTIONS.

After a system transfer function has been entered, selecting Root Locus from the 'Tools' menu gives you the option of redrawing the last Root Locus or overlapping it with a new plot, if a plot has

already been drawn. If not, 'Draw New Plot' can be selected to draw the first plot. If a new plot is to be drawn, you will be asked for various plot values and option selections. The minimum and maximum gain values are input as well as the number of points to plot. The display dimensions are also input at this time. By selecting the max and min values for 'X' and 'Y', any rectangle on the 's' plane can be seen. There is an option for either Linear or Logarithmic point intervals. This determines how the incremental gain values between the max and min will be determined. An example of this is included later in the chapter. The loop path is also an option. This does not change the loop path of the original transfer function. It is only used for plotting the Root Locus. If the transfer function has a nonzero feedback path, then you can select either 'Geq' or 'Closed Loop Path'. These have the same definitions as described in the block manipulator section. Selecting 'Geq' means the gain of the forward path will be adjusted during the plot point calculations. 'Geq' is defined as;

$$Geq = G/(1 + G*H)$$

Where G is the product of the forward path blocks and H is the product of the feedback path blocks. If K is the gain, then the characteristic equation is;

$$1 + K*G*H$$

or

$$Gden*Hden + K*Gnum*Hnum$$

Where 'num' and 'den' denote numerator and denominator polynomials respectively. If 'Closed Loop' is selected, then unity feedback is added to the system transfer function. 'Closed Loop' is defined as;

$$\text{Closed Loop} = Geq/(1 + Geq)$$

The characteristic equation is then;

$$1 + k\,Geq$$

or

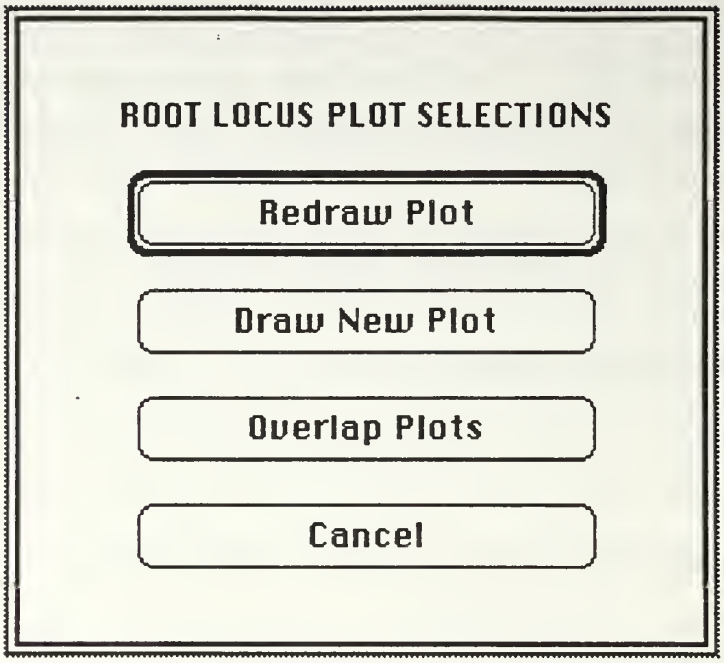
$$Gden*Hden + Gnum*Hnum + K*Gnum*Hden$$

This option allows you to select where to put the gain 'K'. If your system does not have any blocks in the feedback path, you will be notified by an alert box that states the default has been changed from 'Geq' to 'Closed Loop' since selecting 'Geq' without any feedback blocks would not allow the characteristic equation to change as 'K' is changed.

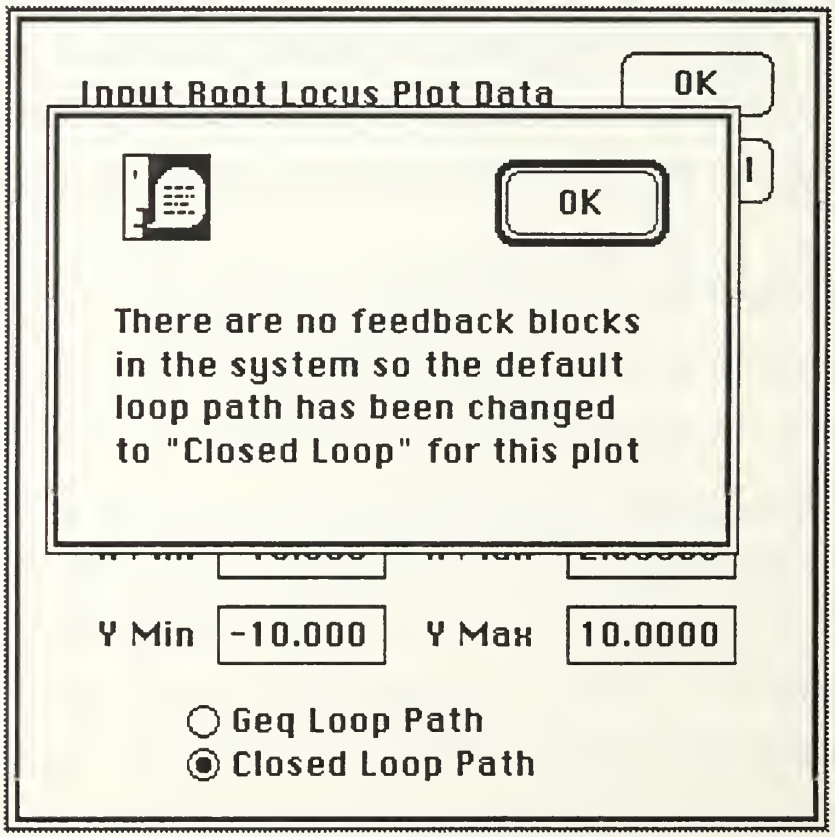
C. FOURTH ORDER CHARACTERISTIC EQUATION EXAMPLE.

As an example of the Root Locus plot and it's options, a one block transfer function has been entered. The numerator is 1 and the denominator was entered in factored form with zeros at -1, -3, -5, and -7. This was selected because it will show how real roots move towards each other in pairs and when they meet, they split and move in opposite directions as complex conjugate pairs. Fig(1) shows the dialog box that is shown after selecting 'Root Locus' from the 'Tools' menu.

In this case, you select 'Draw New Plot' since this is the first plot. You will then see the dialog box as in Fig(2).

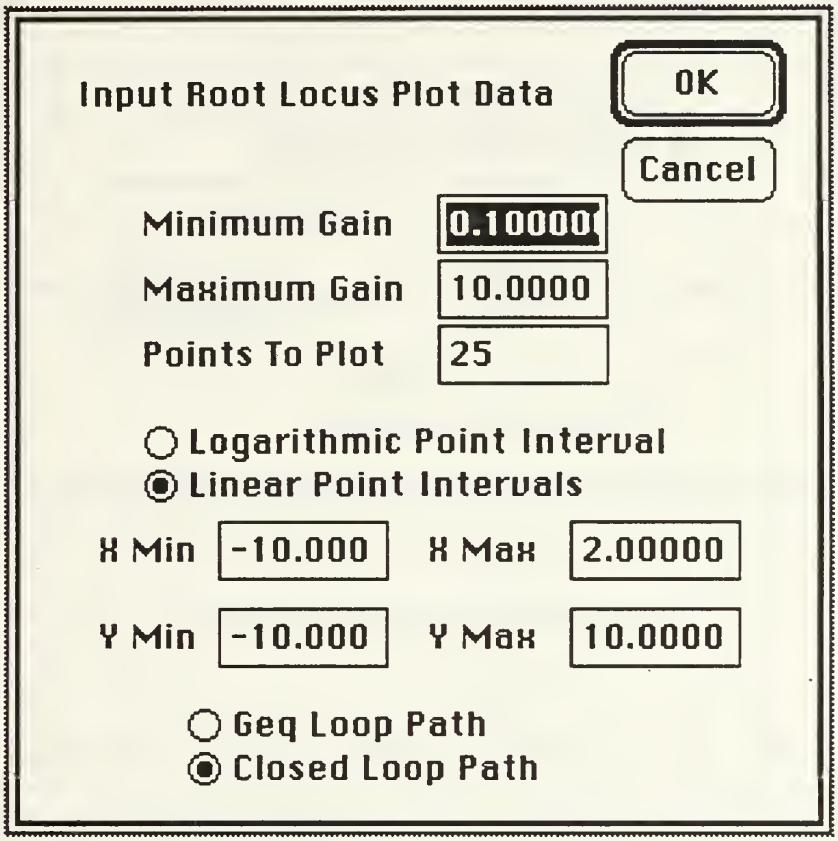


Fig(1) Selecting 'Root Locus' from 'Tools' menu.



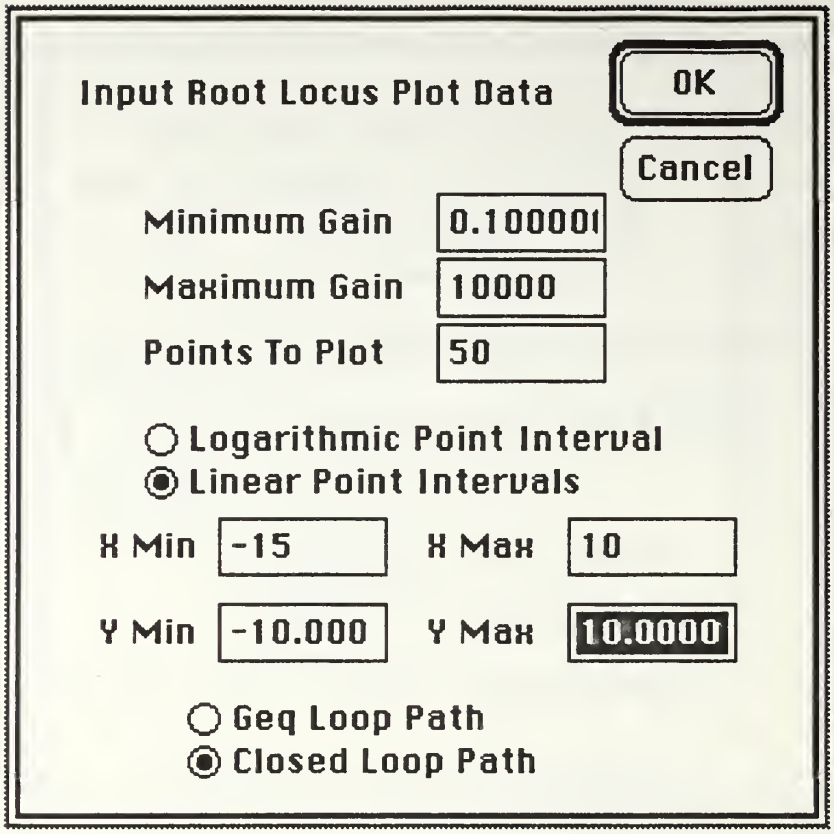
Fig(2) Dialog Box After Selecting 'Draw New Plot'.

Fig(2) also shows the alert box that is displayed since you have not entered any blocks in the feed back path. The alert box must be responded to first so the 'OK' box is selected. The dialog box is then shown in Fig(3).



Fig(3) Root Locus Dialog Box.

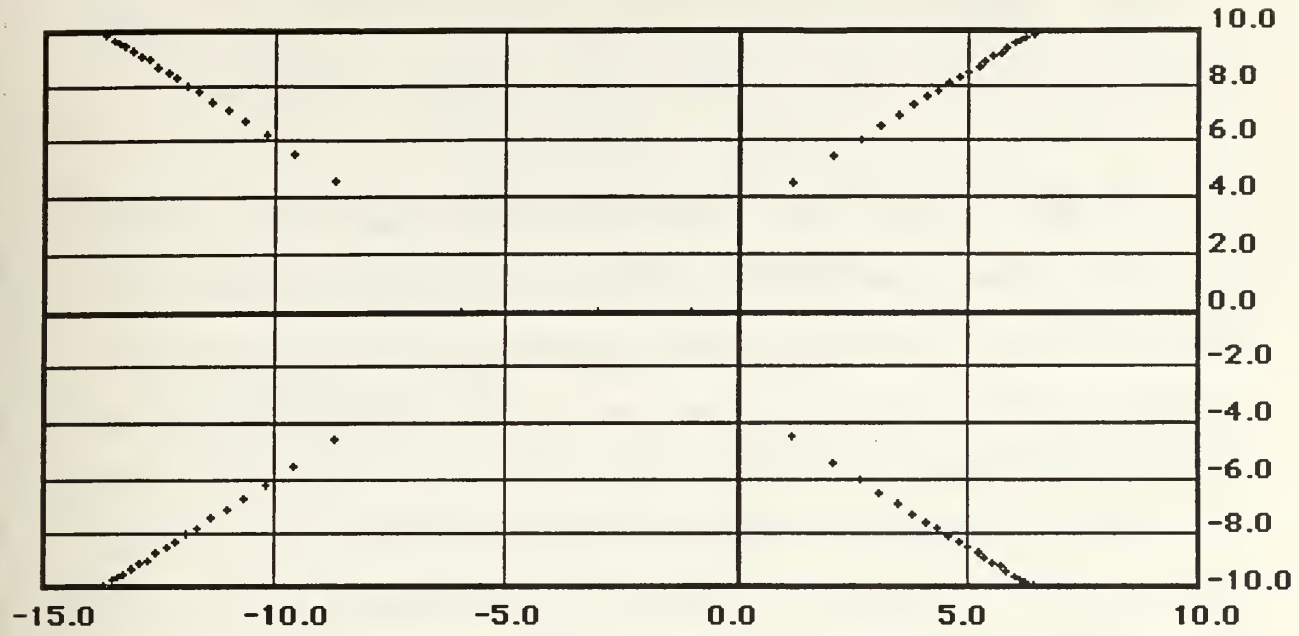
The plot default values are shown in Fig(3). They can be changed as desired and the next time the dialog box is displayed, your last selected values will be shown. For illustrative purposes we will select the values shown in Fig(4).



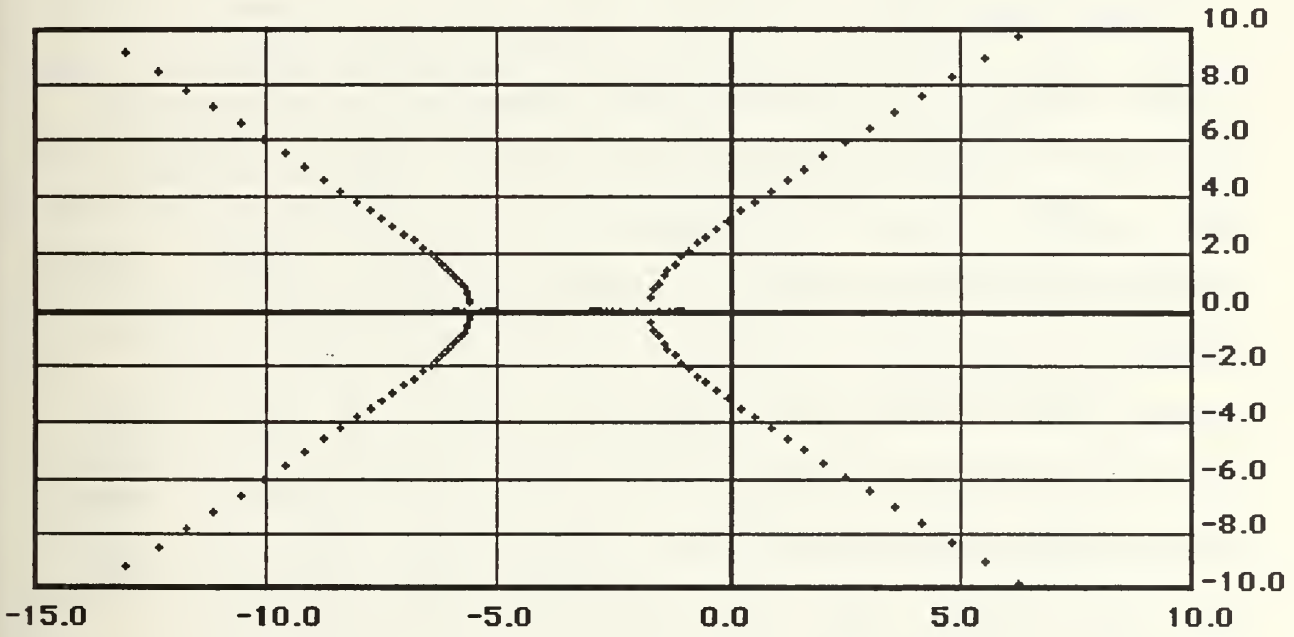
Fig(4) Plot Settings For Example.

The plot will be done twice in order to show the effect of the linear and logarithmic intervals. In the case of Fig(4), we select OK after entering the desired values. An alert box shows the points being calculated and counted down. The plot is then displayed. When enlarged to cover the whole screen, it looks like Fig(5).

For a quick comparison, the plot will be drawn again by selecting 'Draw New Plot' using the same values but selecting Logarithmic Interval. The plot now looks like Fig(6).



Fig(5) Root Locus With Linear Intervals.



Fig(6) Root Locus With Logarithmic Intervals.

The difference in the plots is clear. For 'Linear' interval, the gain step size is calculated by subtracting the min gain from the

max gain, entered in the dialog box, and then dividing by the number of points to plot. In the first plot, Fig(5) the gain step size is;

$$(10000 - .1) / 50 = 199.9$$

The plot in Fig(6) was calculated using 'Logarithmic' intervals. This can be best described by marking the minimum and maximum gain values on a sheet of log paper. Measure the distance between these two points on the log paper using a ruler. For an example, say this distance was 6". Now divide 6" by the number of points to plot. This would be 6/50 = .12" Now with the ruler mark 50 'tics' on the log paper at .12" intervals. Reading the corresponding log chart value at each 'tic' will give the gain values to use for 'Logarithmic' interval calculations.

Most CAD programs calculate the gain intervals using the 'Linear' method. This emphasizes gain values that are closer to the max gain. This becomes more evident as the max gain to min gain ratio increases. Using the 'Logarithmic' interval, gives more emphasis to the lower gains so a more continuous locus can be drawn. As a basic rule of thumb, if the max to min gain ratio is greater than 100, selecting 'Logarithmic' interval will give a more continuous plot. Or, for example, if you are more interested in where the real roots meet and split into conjugate pairs, rather than their exact locations well after they have split, then 'Logarithmic' interval would be best. If you know the roots will split but are more interested in where they are going after they split, than the 'Linear' interval may be better.

As with all the other plots in MacCAD, you can chose to plot a root locus, change a block in the system and then overlap the new plot on the old one. You can overlap as many plots as desired and each new plot[1] will be drawn using a lighter pattern so they can be distinguished. Label boxes can also be added.

D. PROGRAMMER'S NOTES.

The basic algorithm of the subroutine calls for the Root Locus is very similar to that used for the other plots available in MacCAD. Roughly, the user selects Root Locus from the pull down menu or from the keyboard shortcut. He then inputs his desire to redraw an old plot, draw a new plot, overlap plots or cancel the operation. If drawing a new plot, a dialog box is presented where he inputs the plot parameters. The default parameters are either those set during the subroutine 'SetUp' called in the beginning of 'Main' when initializing all variables, or the last values input by the user. 'GetRLocusData' displays the dialog box where the parameters are displayed and entered by the user. The default values are displayed in the dialog box by the procedure 'InitRLocus'. ' GetRLocusData' handles events occurring in the dialog box, such as selecting OK, Cancel or the radio buttons. If OK is selected, then the present parameters are all checked to be the proper type of number. For example, the number of points to plot must be a positive integer and the gains must be positive real numbers with the max gain larger than the min gain.

---

[1] Up to the fourth plot.

Each parameter entered is checked by the function called 'GoodRLocusDataEntered' which returns a boolean 'true' if the parameters are correct and 'false' if not. If a parameter is incorrect, the procedure 'FrameDataError' is called which frames the data box containing the incorrect parameter. If this is the first error detected, then that data box is selected as if double clicked by the cursor. If it is not the first error, the box is just framed. If all data is correct, the procedure 'DrawBasicPlot' is called which draws the plot grid. It calculates the labels and their intervals and also draws them.

The 'DataToGraph' procedure then determines the gain value from the function 'NextGain' and calculates the characteristic polynomial from function 'GetGeq'. This coefficient polynomial is then factored using 'RootFinder'. The factors are then plotted as small crosses through the functions 'PlotPoints' and 'CrossPoint'. A new Geq is calculated for each gain value and the process is repeated until the desired number of points have been calculated and plotted. It should be noted that although selecting a large number of points to plot will give a more continuous locus, the time required to complete the plot would be considerable. 'RootFinder' must be called for each gain value and since it is an iterative procedure with up to 1000 iterations for each complex conjugate pair in the polynomial, it could be very time consuming. This is the reason 'RootFinder' has an option for a lower accuracy, which then requires fewer iterations, and is the only time the lower accuracy option is used in MacCAD.

Accuracy to 3 decimal places would not be noticeable from points plot on a graph.

As with the other plotting routines, the clipping rectangle of the Root Locus window is changed to the rectangle which defines the plot dimensions called 'plotrect'. In this way, if a point is to be drawn which would fall outside of the plot, the clipping region prevents it from actually being drawn.

Overlapping plots is also done similarly to the other plots. The last plot is saved as a picture. 'GetRLocusData' is skipped so the parameters cannot be changed before the next plot. 'DrawBasicPlot' is also skipped since the plot grid and labels have already been drawn by the first plot. Depending on the number of layers drawn, the pen pattern is changed to a lighter shade. 'DataToGraph' is then called which draws only the new points. These are drawn on top of the picture saved of the last plot. The resulting combination picture is then associated back to the Root Locus window.

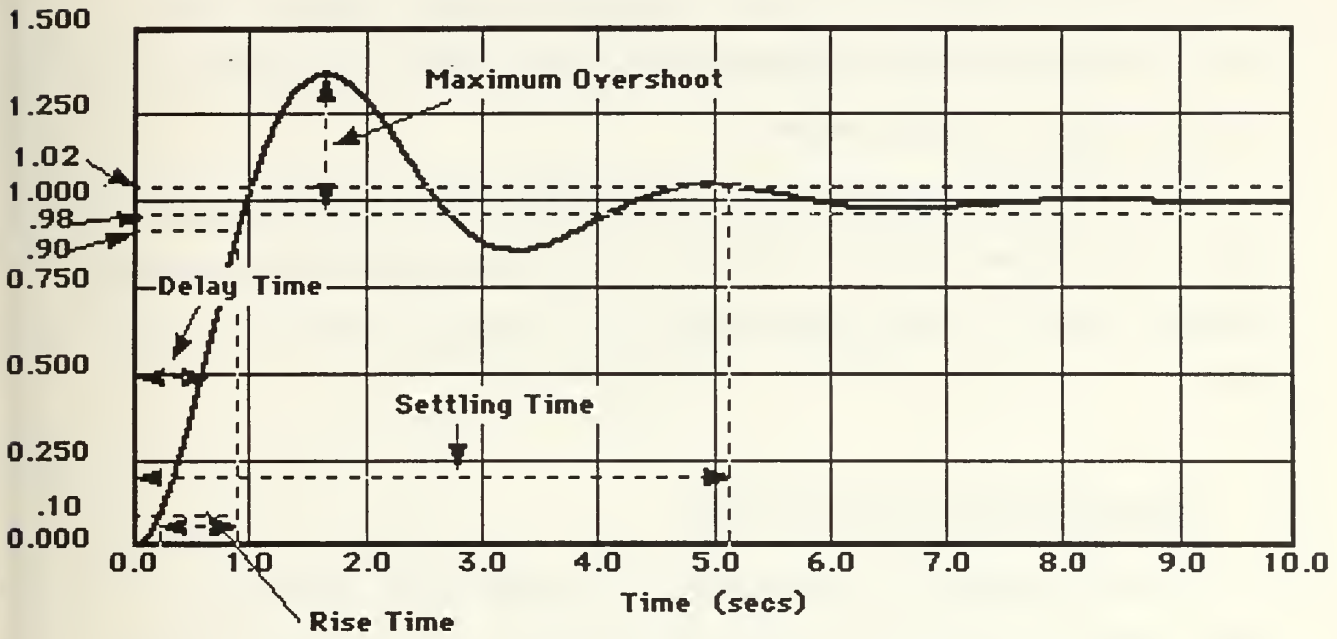
IX. TIME RESPONSE

A. BASIC DESCRIPTION.

The 'Time Response' item under the 'Tools' menu allows the user to view the output of the system in the time domain using one of a variety of standard inputs. These inputs include the step function, ramp, impulse and sine wave. As with the other plots available in MacCAD, there is always the option of overlapping plots or adding labels.

The Bode, Nyquist and Root Locus plots can all be used to determine the systems stability in addition to a great deal of other information. The stability of the system may be easier to visualize with the time response plots. The standard time domain input used for determining stability is the unit step function. From the plot of the unit step time response you can measure several characteristics of the systems transient response such as the rise time, delay time, maximum overshoot, settling time and the steady state error. Rise time is the time it takes the output to increase from 10% to 90% of it's final value. Delay time is the time required for the output to reach 50% of it's final value. The maximum overshoot, usually expressed as a percentage of the input value, is the difference between the largest value the output reaches and the input value. This only applies if the outputs largest value is greater than the input value. Settling time is the time it takes for the output to reach

and stay within 2% of its steady state value. The steady state error is the difference between the input value and the output value as time approaches infinite. Fig(1) shows some of these quantities.



Fig(1) Transient Response Characteristics.

B. USER OPTIONS.

When 'Time Response' is selected from the 'Tools' menu, the user has the option to redraw the last Time Response plot, draw a new plot or overlap a new plot on the last one. Drawing a new plot is done by selecting the type of input function. If the Step function is selected, the default step amplitude is 1.0 but it can be changed to any real number. The Ramp function has a user adjustable slope and an optional D.C. offset. The Sinewave function has an adjustable frequency as well as amplitude. The Impulse has an adjustable

amplitude but also has the option of automatically setting the amplitude so the input will be a unit impulse.

The plot dimensions are input as a maximum plot value and the maximum time to plot. There is also the option to place the zero value of the ordinate at the plot's center or at the bottom of the plot. This option is not available for the sine wave input which always places it at the center.

As with other MacCAD plots, the overlap option draws a new plot on top of the last one, using the same plot parameters as the previous plot. The same input function will also be used for the new plot.

C. TIME RESPONSE CALCULATIONS.

The time response is calculated by changing the continuous linear system transfer function into discrete state-space matrices. This is done by first describing the system states with the familiar matrices;

$$\underline{x}'(t) = \mathbf{A}\,\underline{x}(t) + \mathbf{B}\,u(t)$$
$$\underline{y}(t) = \mathbf{C}\,x(t)$$

The **A**, **B** and **C** matrices can be determined from the original transfer function.

$$\frac{a\,s^2 + b\,s + c}{s^3 + d\,s^2 + e\,s + f}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -f & -e & -d \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} c & b & a \end{bmatrix}$$

These continuous matrices are then transformed into the discrete time matrices;

$$\underline{x}(k+1) = \Phi\ \underline{x}(k) + \Gamma\ \underline{u}(k)$$

$$y(k) = C\ \underline{x}(k)$$

using the definitions;

$$\Phi = I + A\ \mu(T)$$

$$\Gamma = \mu(T)\ B$$

with T equal to the sampling period and μ defined as;

$$\mu(T) = \int_0^t e^{A\sigma}\ d\sigma = T \sum_{k=0}^{\infty} (A^k\ T^k)/(k+1)!$$

The actual value of T is determined by the max time parameter input by the user. The max time value is divided by 1000 to get the sample interval T. The calculation of these matrices will be discussed further in the Programmer's Notes section.

D. ILLUSTRATIVE EXAMPLE 1. THE UNIT STEP INPUT.

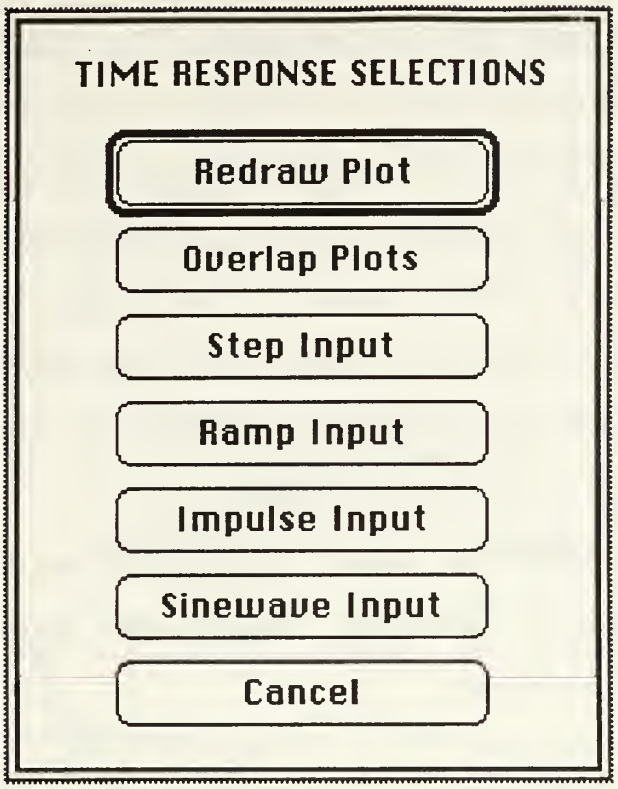
A lightly damped second order system will be used to illustrate the Time Response, starting with the unit step. The transfer function is given as;

$$\frac{\omega n^2}{s^2 + 2\zeta\omega n + \omega n^2}$$

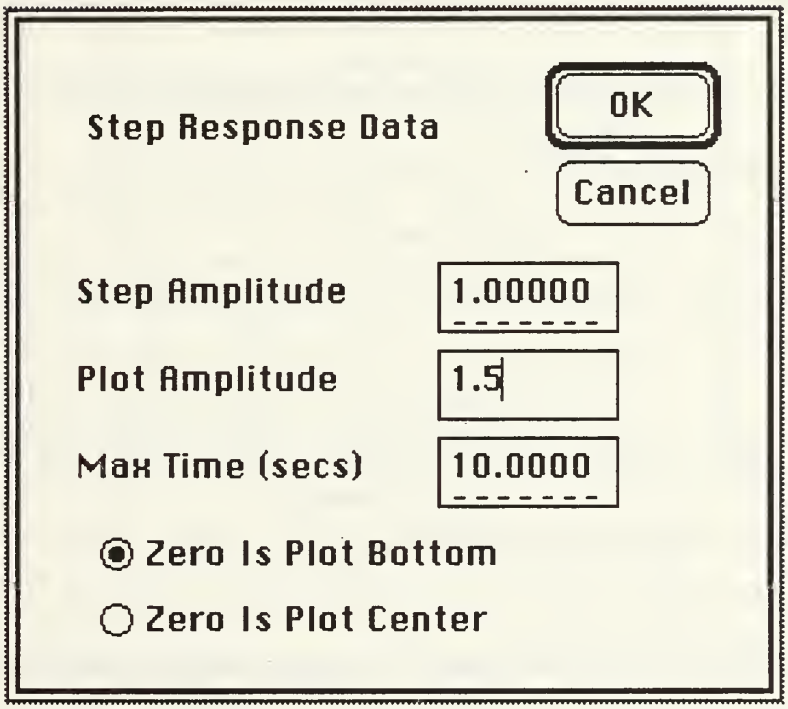
where $\omega n = 2$ and ζ will be varied from .25 to 1.0 in order to show it's effects. Starting with the smallest value for ζ the following transfer function is input as block #1.

$$\frac{4.0}{s^2 + 1.0\,s + 4.0}$$

The 'Time Response' item from the 'Tools' menu is selected and the dialog box displayed is shown in Fig(2). This gives the redraw and overlap options along with the various input functions for a new plot. The Step Input will be selected at this time. The Step Input dialog box is shown in Fig(3).



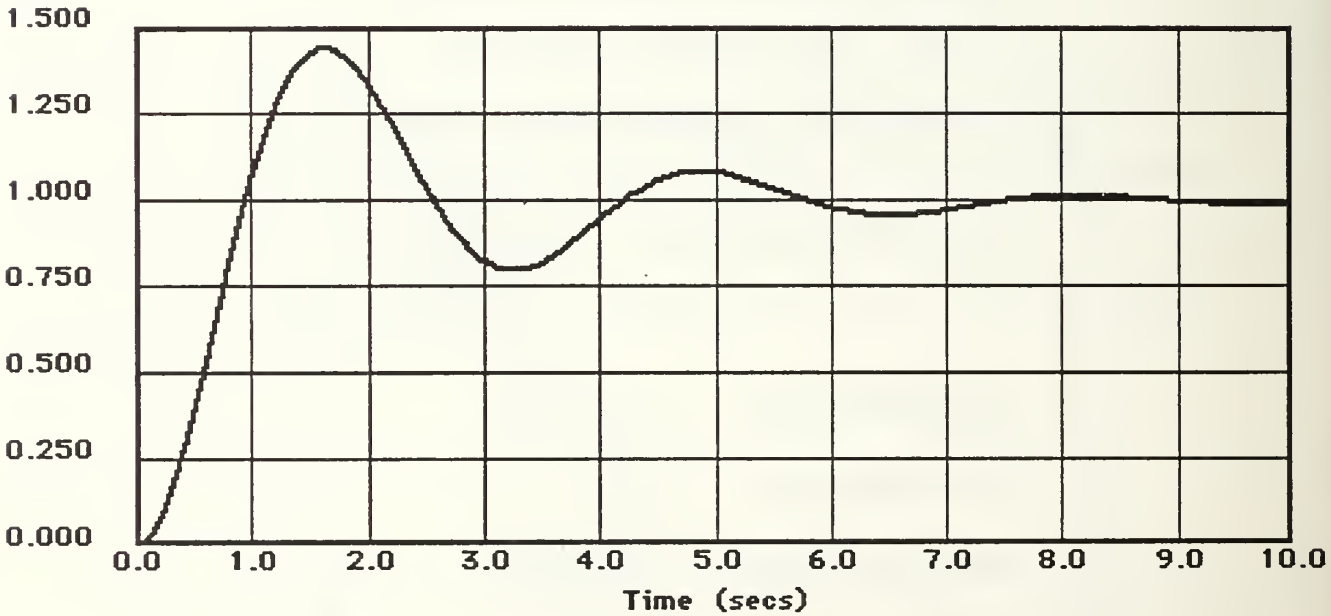
Fig(2) Time Response Dialog Box.



Fig(3) Step Input Dialog Box.

The only parameter changed was the Plot Amplitude. It was increased from the default value of 1.25 to 1.5 because the system is lightly damped and we know there will be overshoot. The Step Amplitude is left at 1.0, making this the unit step response. The radio button by 'Zero Is Plot Bottom' is left at the default value because the system should not have such large oscillations that they would go below zero. For very lightly damped or marginally stable systems, this option might be selected.

After 'OK' is selected, an alert box counts down from 1000 to 0 as each time sample is calculated. The plot that shows the unit step response is shown in Fig(4).

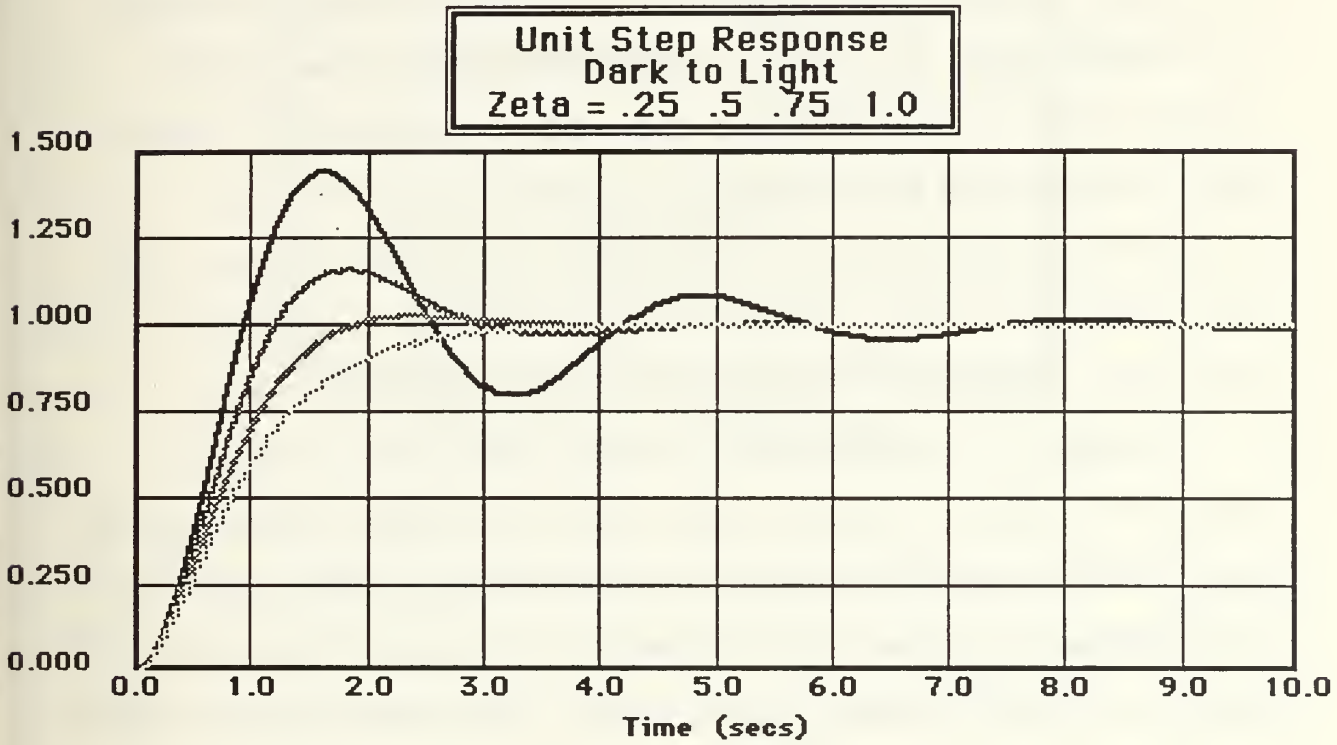


Fig(4) Unit Step Response ζ = .25

From Fig(4) the max overshoot is approximately 40%, the delay time about 1/2 seconds as is the rise time. The settling time is

about 7.0 seconds if you consider the 4th oscillation being within 5% of the final value.

To show how ζ affects the response, it will be changed to .5, .75 and 1.0 The plots will be overlapped. The final step response is shown in Fig(5).



Fig(5) Final Overlapped Plot Showing Various Values For ζ

E. ILLUSTRATIVE EXAMPLE 2. THE RAMP INPUT.

MacCAD also has the capability to examine the transient response to a ramp input with an optional D.C. offset, or a sine wave. Unlike for the unit step input, the sine or ramp responses can be difficult to analyze without the input also being plotted on the same graph. The Overlap option is ideal for this situation. For this

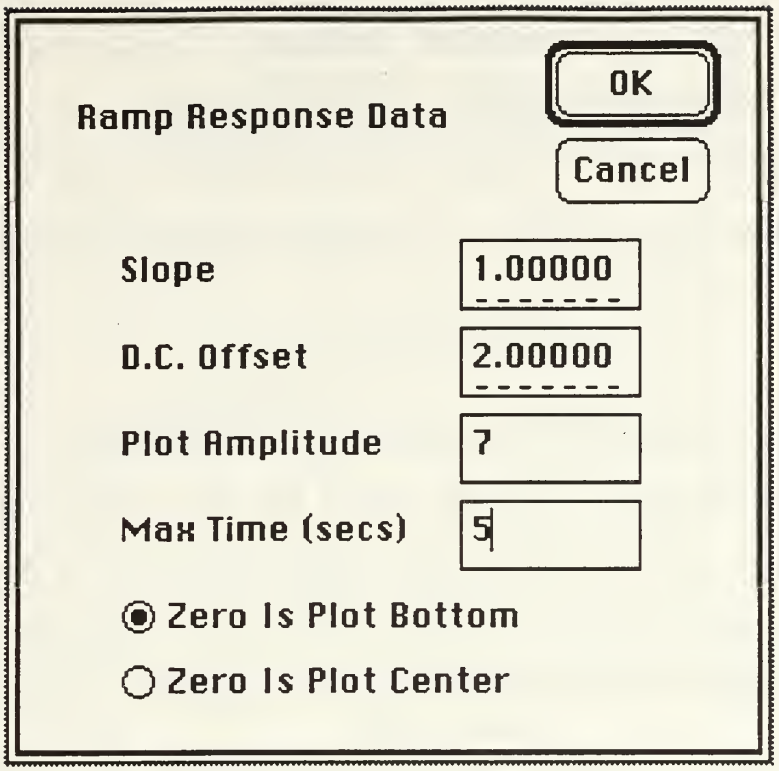
example, the second order system of the previous example will be used. ζ will be set to .25 so the system will be lightly damped. Once the system transfer function has been set as desired, the block data is saved to a file.[1] After this is done, a new transfer function is loaded. It is a simple function but it may be desirable to also save it to a file called 'Unity Function' perhaps because it may be used often. The Unity Function has a transfer function which equals a constant. It is entered using the 'Blocks' menu items as any other transfer function. Enter zero for both the numerator and denominator order. Enter 'ones' for the gain constants and the s^0 terms for both numerator and denominator. This makes the transfer function = 1/1. The Bode plot of this transfer function would give a straight line along the 0 dB line and 0 phase over the entire frequency range.

Selecting 'Time Response' for this function will obviously display the input on the plot since when the transfer function equals one, the output equals the input. Using the Unity Function, the ramp response will be plotted. For this example, a slope of 1.0 and a D.C. offset of 2.0 will be used. After selecting 'Time Response' from the 'Tools' menu and 'Ramp Input' from the dialog box shown in Fig(2), the dialog box shown in Fig(6) is displayed.

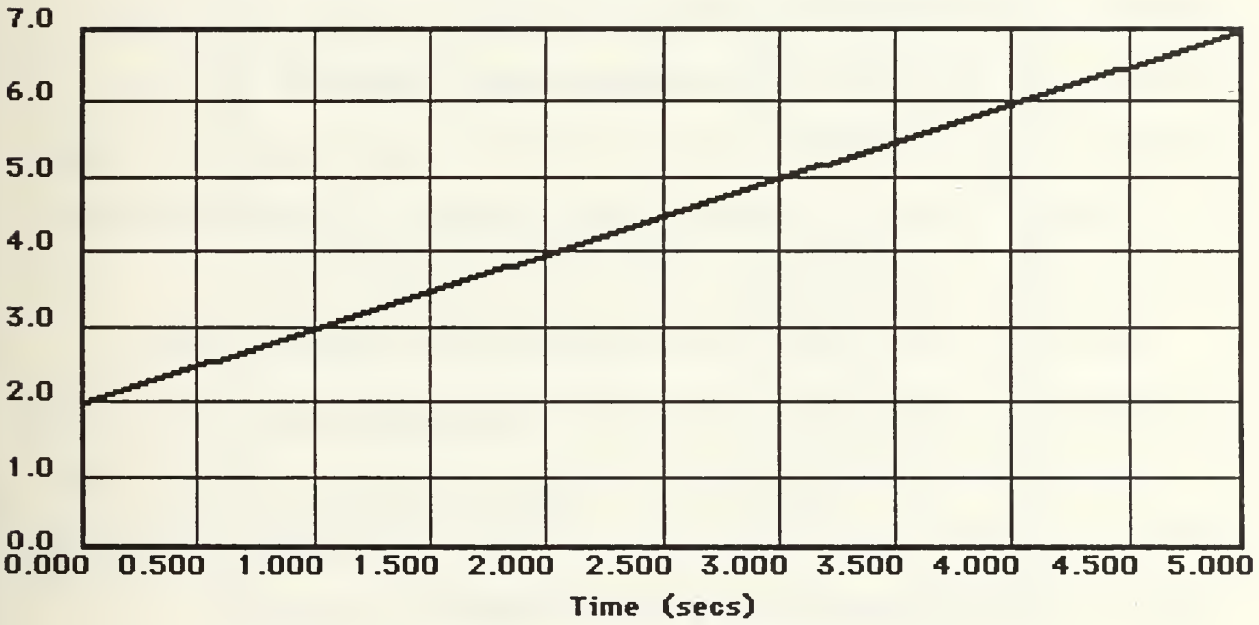
The offset and slope values are set as described earlier. We want to see the first 5 seconds of the output so the Plot Amplitude is set to 7.0 in order to show the output, or the input in this case, over the entire time period selected. After clicking 'OK' the plot is shown in Fig(7).

---

[1] See section covering Standard Macintosh Menus.



Fig(6) Ramp Input Dialog Box.



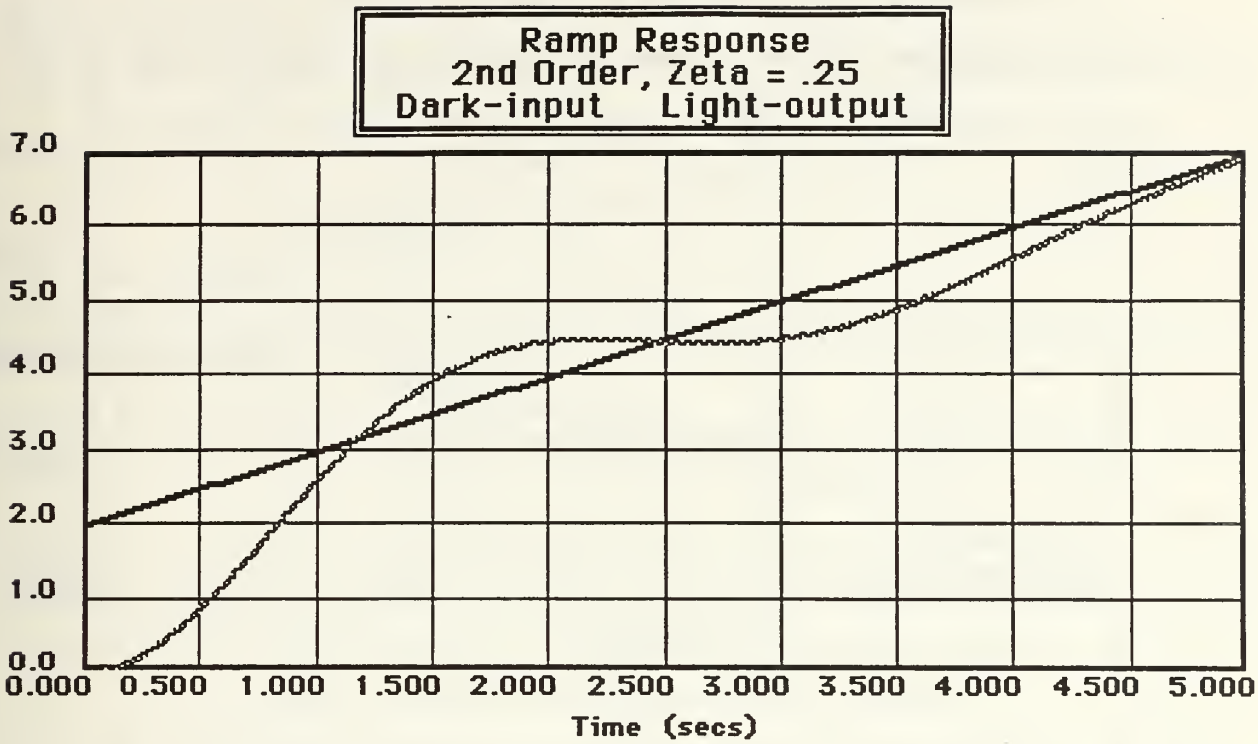
Fig(7) Unit Ramp Input With D.C. Offset of 2.0

The plot now shows the input signal. The second order system is now loaded back into the system block and 'Overlap Plots' is selected from the 'Time Response' dialog box. The resulting overlapped plot is shown in Fig(8). It can be seen from this plot that the output overshoots the input at time = 2 seconds and then stabilizes at 5 seconds.

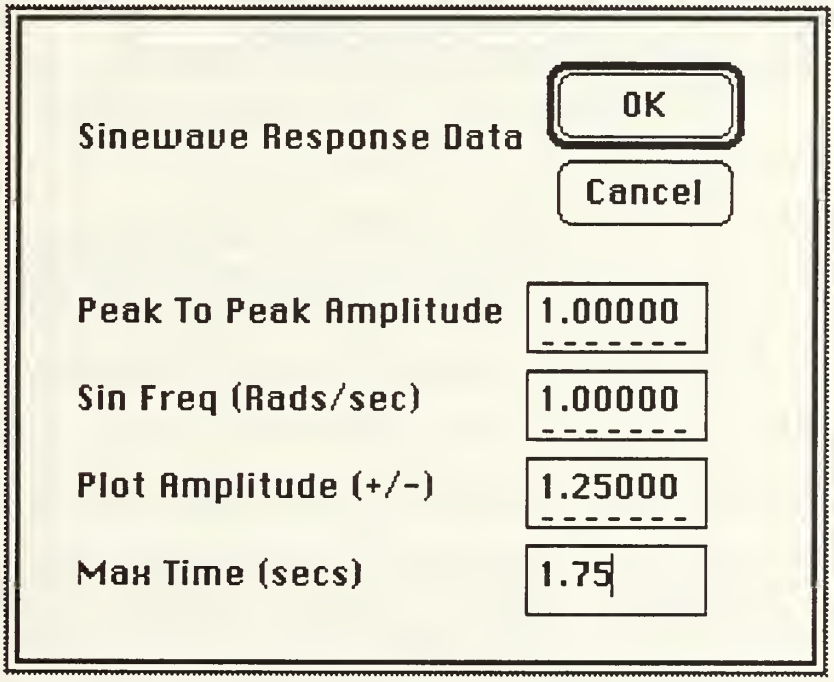
F. ILLUSTRATIVE EXAMPLE 3. THE SINE WAVE INPUT.

This example will show the use of the sine wave input as well as another use of the overlap option and the Unit Function. A servo which is described by the second order transfer function used in the previous examples will be used here. The input to the system will be a reference signal in the form of the first quarter of a period of a sine wave with a radial frequency of 1.0. It will be directing the servo to move from it's present position, the origin, to the desired position, 1 unit away. It is desired to determine when the output position has less than a 20% position error relative to the input reference signal while en route to the final desired position located 1 unit away. This will be done using a variation of the Unit Function and the 'Sine wave Input' from the 'Time Response' tool.

The Unit Function and overlap option will be used to draw an 'envelope' with less than a 20% error. After loading the Unit Function into the System block, the numerator gain constant is changed from 1.0 to 1.2. Now select 'Sine wave Input' from the dialog box shown in Fig(2). The next dialog box is shown in Fig(9).

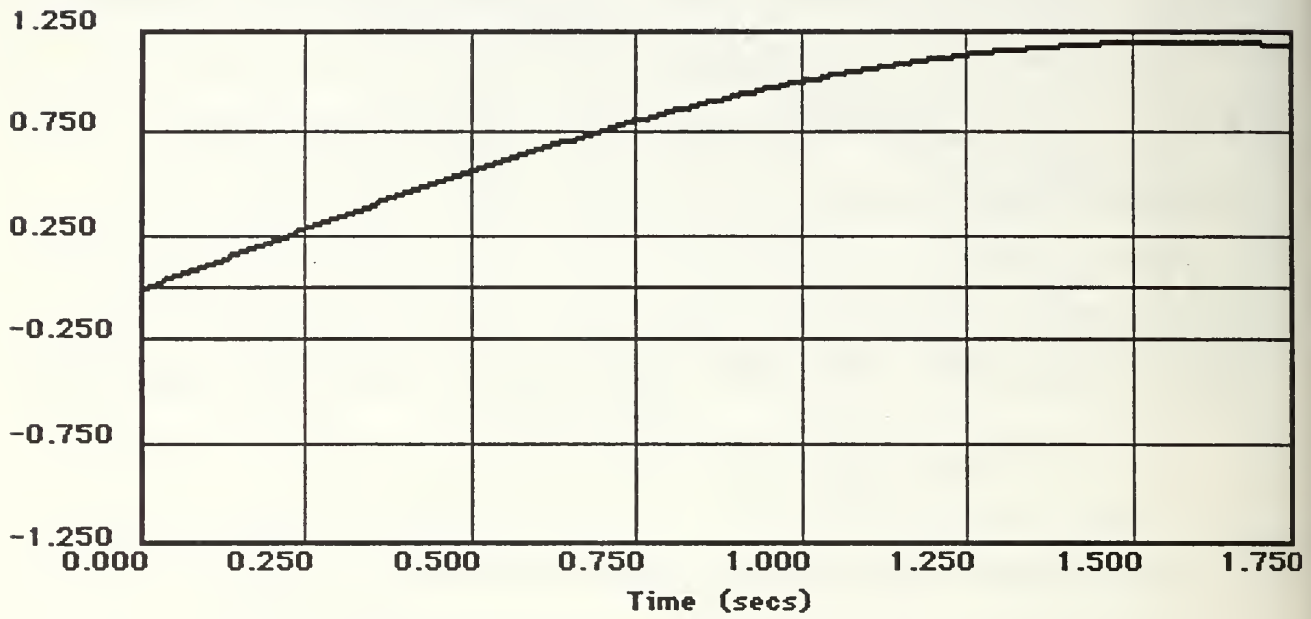


Fig(8) Ramp Input And Output With ζ = .25



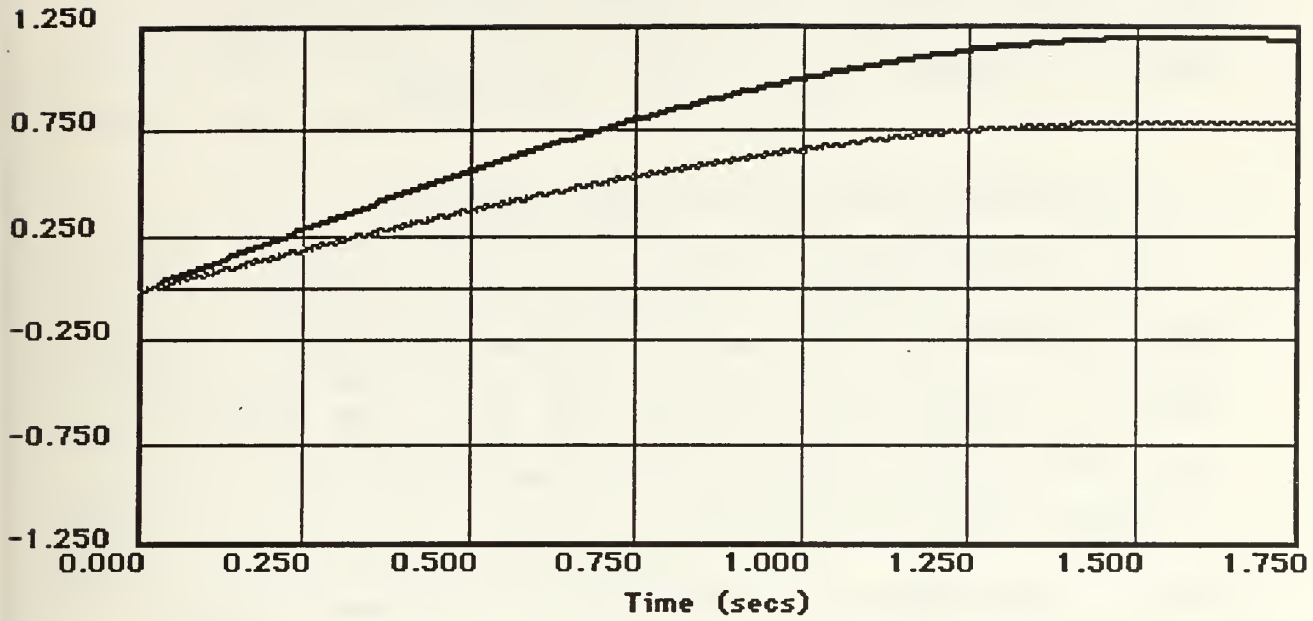
Fig(9) Sine Wave Input Dialog Box.

The frequency has been set to 1.0 and the Max Time has been set to 1.75 because this will show the first quarter period which is all we are interested in. After clicking on 'OK' the plot is shown in Fig(10).

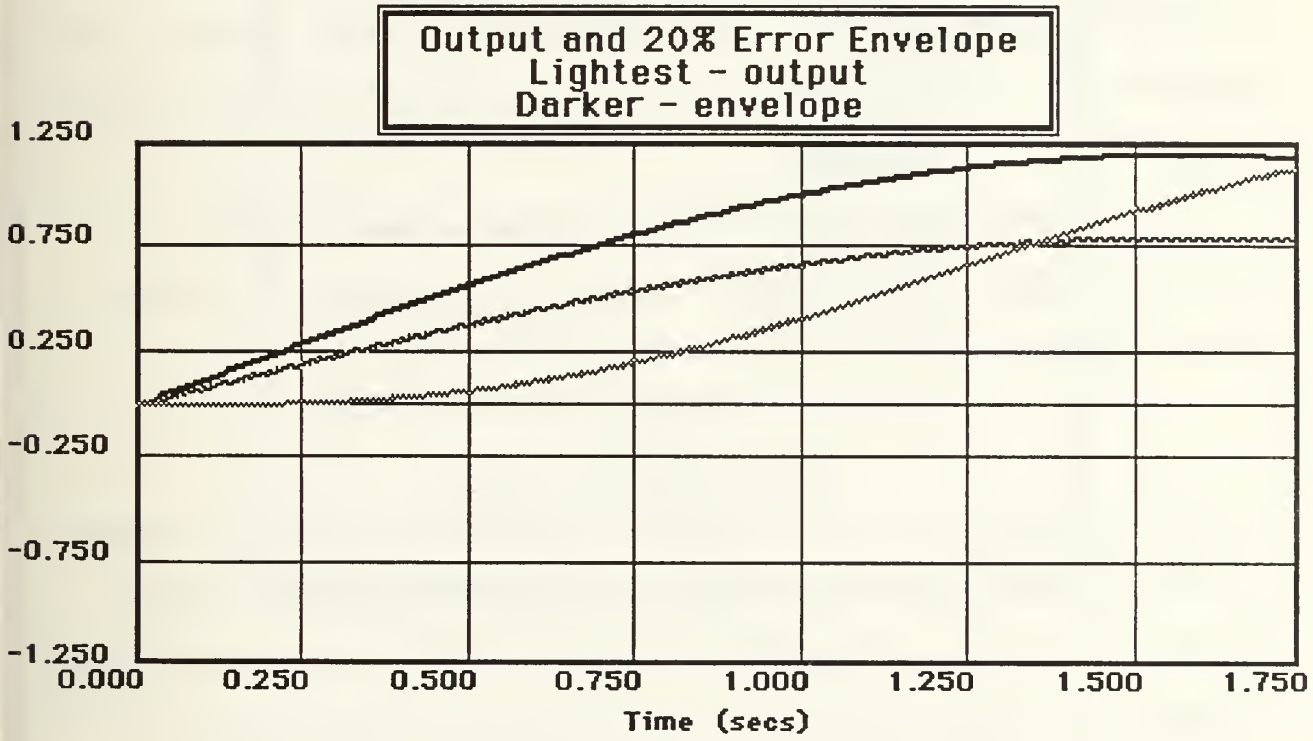


Fig(10) Upper Limit Of 20% Output Error Plot.

Since the numerator gain constant of the Unity Function was changed to 1.2, the resulting plot is 1.2 times a sine wave of amplitude 1. This forms the upper limit of the 20% output error. The numerator gain constant is now changed from 1.2 to 0.8 and overlapping a new plot will now draw the lower limit of the output error. Fig(11) shows the completed plot of the 20% error envelope. The second order system can now be loaded and the 'Time Response' overlap option again selected. The final plot is shown in Fig(12).



Fig(11) Plot Of 20% Output Error Envelope.



Fig(12) Final Plot Of Output And Error Envelope.

The plot clearly shows that even though the output is nearly 1.0 after the quarter period, 1.57 seconds, it does not decrease the position error below 20% until time 1.35 seconds.

G ILLUSTRATIVE EXAMPLE 4. THE IMPULSE INPUT.

This example will use some simple Laplace transforms and Laplace definitions to illustrate the Impulse Input option. We will describe the system transfer function as G(s). If we excite this system with an input, X(s) then we can describe the output, Y(s) = X(s) G(s). If X(s) is the Laplace of the input signal in the time domain, then the output signal in the time domain is the inverse Laplace of Y(s). This procedure is simplified if the input is a unit impulse in the time domain which has a Laplace transform of 1. This makes X(s) = 1.0 which implies that Y(s) = G(s). This means that the output in the time domain, is the inverse Laplace of the system transfer function G(s).

To illustrate this using MacCAD, we will obtain the time response using a unit impulse input for a few transfer functions, G(s). The first transfer function will be in the form of;

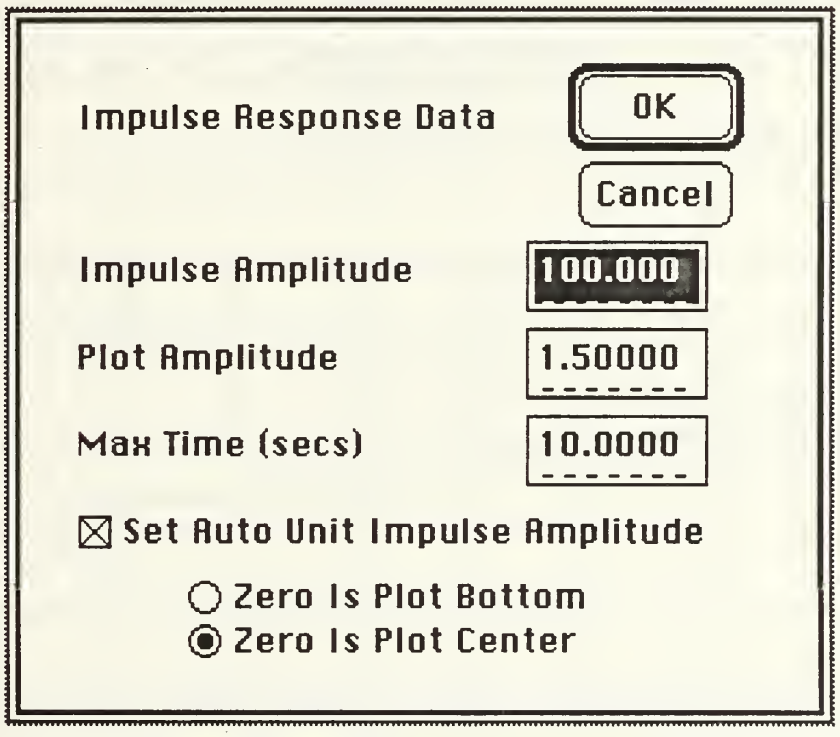
$$\frac{1}{(s - a)}$$

which is recognized as the Laplace transform of e^{at}. We will select 'a' to be -0.25 so the exponential will decay and the time constant will be 4. The settling time should be about 3 * 4 = 12 seconds. This makes the transfer function;

$$\frac{1}{(s + .25)}$$

This transfer function is entered into the System block. 'Impulse Input' is then selected from the 'Time Response' dialog box of Fig(2). The following dialog box is shown in Fig(13).

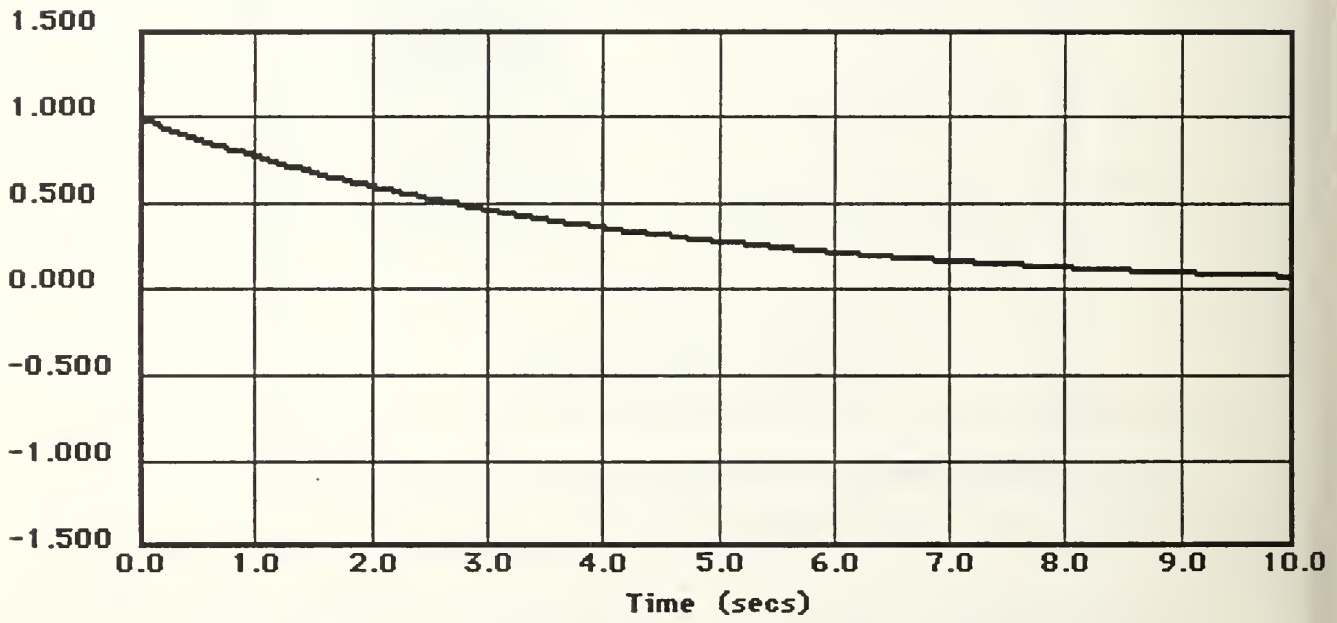
A new option appears on this dialog box. It is the 'Set Auto Unit Impulse Amplitude' check box. This obviously applies only to the impulse input. Since the time response is calculated discretely, and the impulse is approximated as a square pulse with a width equal to the sampling interval T described earlier, the amplitude must be adjusted to be 1/T in order to make the area under the 'impulse' curve equal to 1.0 for a unit impulse.



Fig(13) Impulse Input Dialog Box.

As 'Max Time' is changed, the value of T will change, thus requiring the impulse amplitude to also change in order to maintain

it's unity value. Checking the check box lets MacCAD do this calculation automatically each time the Max Time is changed. This can be seen by entering a new number in Max Time and seeing the 'Impulse Amplitude' change. As each digit is added or changed to Max Time, the Impulse Amplitude changes. Clicking the mouse in the check box will remove the 'x' signifying that the option is deselected. This allows the user to enter any amplitude desired and it will not be changed if the Max Time is changed. In our example, the 'unity' area is desired so the check box is left at the default setting indicating the option is activated. We will also click the radio button changing zero to the plot center. The need for this will be evident later in the example. After selecting 'OK', the time response is displayed as in Fig(14).

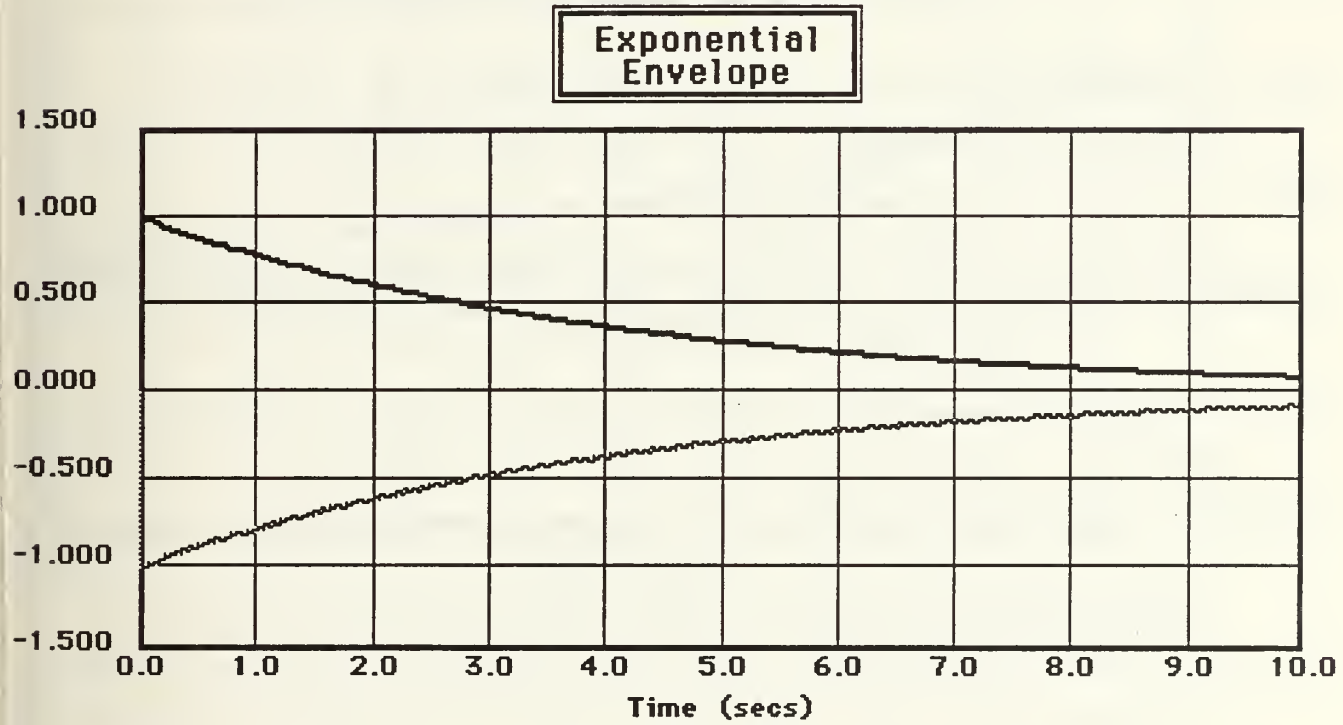


Fig(14) Unit Impulse Response Of 1/(s+.25)

The time response correctly displays the exponential decay which is the inverse Laplace transform of the transfer function G(s) = 1/(s + .25) We will change the numerator gain constant from 1.0 to -1.0 and overlap the new plot. This will make a mirror image of the first plot forming the exponential envelope as in Fig(15).

As a final illustration, an new transfer function will be entered. This will be in the form of;

$$\frac{s-a}{(s-a)^2 + b^2}$$

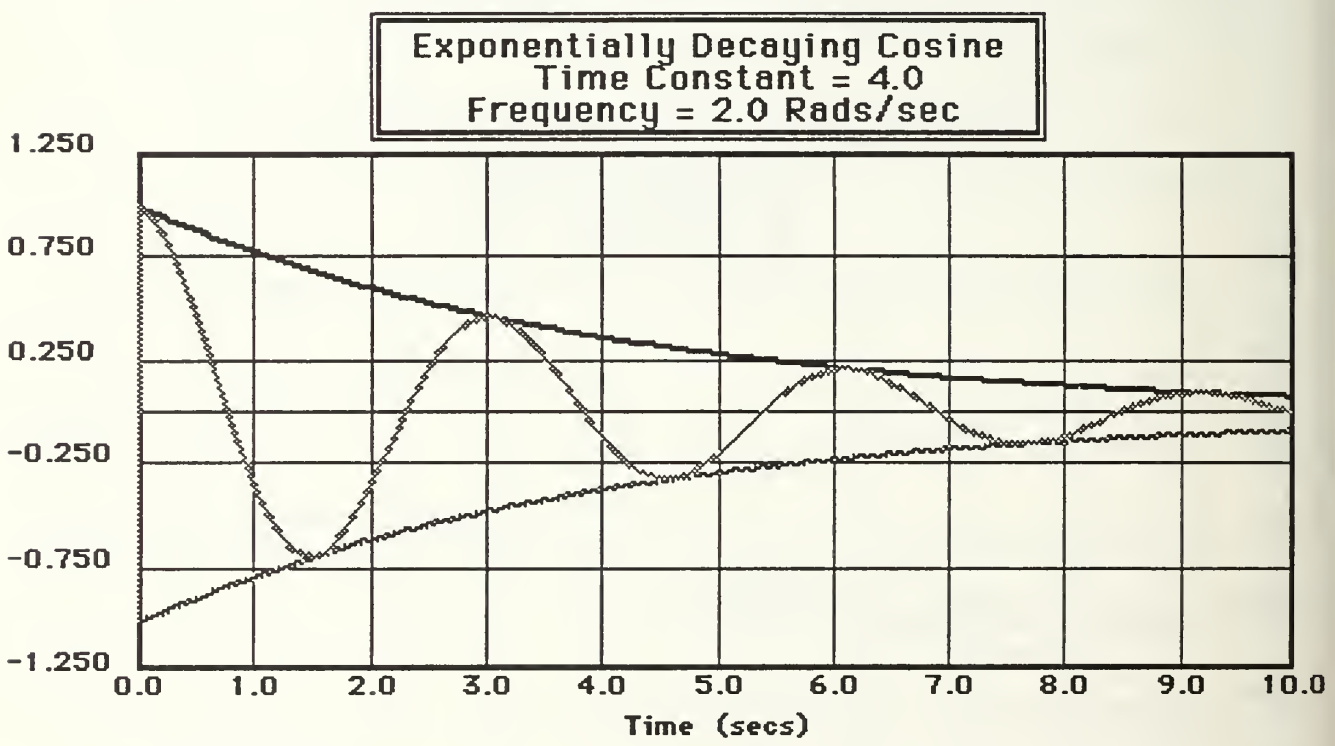


Fig(15) Overlapped Plot Forming Exponential Envelope.

which is the Laplace transform of e^{at} cos bt. Setting 'a' = -.25 and 'b' = 2.0 makes G(s) equal;

$$\frac{s+.25}{(s^2 + .5s + 4.25)}$$

The new G(s) is entered as the System block and overlap plots is once again selected from the 'Time Response' dialog box. The resulting plot is shown in Fig(16).



Fig(16) Final Overlapped Plot Showing Exponential Decaying Cosine.

H. PROGRAMMER'S NOTES.

A variable called 'timedata' is used to store the information about the plots for the various types of inputs. All the parameters

for this and all other plots are initialized when MacCAD is first started in the module CAD SetUp. The 'timedata' variable is defined as;

```
timedata : RECORD
    inputtype  : integer;
    maxtime  : extended;
    amp : extended;
    impamp : extended;
    autoimpamp : boolean;
    zerobottom : boolean;
    freq : extended;
    slope : extended;
    dcoff : extended;
    maxy : extended;
    layer : integer;
    doit : boolean;
```

'inputtype' is an integer identifying the type of input used. 1 is for step, 2 for ramp, 3 for impulse and 4 for sine wave. 'maxtime' corresponds to the plot parameter 'Max Time' input in the dialog boxes prior to plotting. 'amp' is the amplitude of the inputs. 'impamp' is the input amplitude for the impulse input. The boolean 'autoimpamp' is true when the impulse amplitude is automatically set based on the 'Max Time' input by the user. Boolean 'zerobottom' is true if the plot zero is to appear at the plot bottom and false if it is to appear at the center. 'freq' is used only for the sine input, as the radial frequency of the sine wave. 'slope' and 'dcoff' are used for the ramp input and is the slope and D.C. offset. 'maxy' is the max plot amplitude. 'layer' is used to identify how many plots have been overlapped. This is used to determine the pen pattern of the next plot. The boolean 'doit' is used during the input dialog boxes. It is

used both as a flag indicating the user does want the new plot drawn and for saving the input parameters as the new timedata.

Some simple procedures and functions within the CAD Time Menu module include 'Mag2Ht' which returns the vertical pixel position corresponding to the height value input relative to timedata.maxy. 'Time2Wd' converts a value of time input to a horizontal pixel position based on timedata.maxtime. 'MatrixMult' multiplies two square matrices of the same order with the matrix names and the order as input parameters. The function returns the resulting matrix product. The functions 'ScalarMatrixMult' uses two nested 'for' loops to multiply all the elements of an input matrix by an input scalar and returns the new matrix. 'MatrixVectorMult' multiplies an input matrix by an input vector and returns the resulting vector. There is a procedure/function pair for each input type that displays the corresponding dialog box and checks for proper responses. For example the step input uses the 'GetStepData' procedure and the 'GoodStepDataEntered' function as it's check. Due to the differences between the data required to be input by the user for each input type, it was not practical to make a generic procedure/function pair that would work for all four input types.

The procedure 'CalculateMatrixAndVector' uses the above mentioned matrix manipulation functions to approximate the μ matrix which is then used to calculate the Φ and Γ matrices. 'CalculatePlotPoints' then uses the discrete matrices to calculate the state values at each time step. A case statement using

'timedata.inputtype' is used to calculate the input value at each time step.

Since there are 1000 calculations and the plot is only about 300 pixels wide, each calculation need not be sent to the screen. The variable 'timetodraw' holds the value of time that the last point was sent to the screen. It is incremented by an amount of time corresponding to 3 pixels on the screen. 'plottime' is the variable holding the value of time used for each of the 1000 calculations stepped by T. When 'plottime' increases above 'timetodraw' the latest value of the output is then calculated and plotted on the screen and 'timetodraw' is incremented by the 3 pixel value.

One of the first procedures called prior to calculating the data points is 'CalculatePlotDimensions'. This procedure calculates 'delt', the interval used with 'timetoplot' and 'timeinterval' which is the sample time T. The intervals 'plotmagstep' and 'plottimestep' are calculated using the 'FindSep' and 'FindRealSep' procedures in 'NumberCrunch' module. These intervals are the steps between labeled points for the 'y' and time axes on the plot.

As with the other plot modules in MacCAD, the CAD Time Menu module has the procedure 'DrawBasicPlot' which sets the clip rectangle to the plot size and saves the Quickdraw graphics as a picture associated with the 'timePtr' window. 'DoHorizGrid' and 'DoVertGrid' draw the horizontal and vertical grid lines of the plot and 'DoDataPlot' actually draws the time response curve. This is also the procedure called when overlapping plots as it does not affect the plot grid or labels already drawn.

'DoTimeMenu' is the only exported procedure in this module. It is called by the 'CAD Menu Handler' module when the 'Time Response' item is selected from the 'Tools' menu. This is the procedure that displays the Time Response Dialog box, which is actually an alert box, that lets the user select Redraw, Overlap or one of the input types for a new plot. Depending on the response from the alert box, either 'DoDataPlot' is called, in the case of overlapping plots, or the procedure/function pair associated with the desired input type.

As with the other plot modules in MacCAD, the plot dimensions (in pixels) and other numbers such as the total number of calculations to make, are contained as global constants to the module. This allows them to be easily changed for future modification.

I. USERS' TIPS.

A potential stumbling block for the user is forgetting to change the loop path before examining the time response. If the Nyquist plot or open loop Bode plot has just been checked, be sure to check the loop path prior to getting the time response. This applies to using any of the plots in MacCAD. The user should always know what the current loop path of the system is before drawing any new plots.

When checking the unit step response, the output may not be approaching a final value of 1. This could be foreseen by checking the System block's D.C. gain. This is done by multiplying the numerator gain constant and s^0 term together and dividing by the

product of the denominator gain constant and s^0 term. If this resulting value is not 1.0 then the output will probably not approach the same value as the input. This applies to ramp and sine wave inputs as well. The D.C. gain should be checked before drawing the input signal on the plot as described in examples 2 and 3. Checking the D.C. gain and adjusting the Unity Function appropriately lets the input signal be adjusted in amplitude for easier comparison with the output. For example, a sine wave input with a zero to peak amplitude of 1.0 may yield an output with an amplitude of .5. By adjusting the Unity Function before plotting the 'input' to make of the same amplitude as the output makes it easier to compare the output with a true sine wave.

The default plot parameters are set each time a new plot is made. The option to have zero at the center or bottom is carried from one plot to another as is the plot and input amplitudes and max plot time. This is done as a convenience when repeatedly drawing the same plot but could cause confusion when changing input types. Be sure to check all the plot parameters before selecting 'OK' to avoid unnecessary plotting. The sine wave input does not change the default setting of the zero location and the impulse input does not change the input amplitude for any other input type plot.

Since 1000 points are calculated, regardless of the max time to be plotted this means that the relative accuracy will decrease. Displaying a sine wave input of a high frequency over a long period of time may not look like an exact sine wave since it is sampled. If 100 seconds is to be displayed, then the sample interval, T is .1

seconds. If the input has a large ramp slope or sine wave frequency, there could be large changes in the input between each sample.

When selecting the max plot value and max time it is best to use as round numbers if possible. If an obscure max time such as 1.57 seconds is input, it may not be nicely dividable to give time steps on the plot. If this is done, the max time may be rounded up to the nearest 10% and step intervals may not be found. For example, if the value of 1.57, being a quarter of a period of a 1 rad/sec sine wave was input in example 3, it would have been rounded up to 1.6 and no intervals would be displayed. Instead the value 1.75 was used which was easily divided and a time interval of .25 was used for plot labeling. If the time plot ever displays time or 'y' labels that do not meet your needs, try making small adjustments in the Plot Amplitude or Max Time parameters input.

X. MACCAD SUBROUTINES AND LIBRARIES

A. BASIC DESCRIPTION.

This section will basically be Programmers' Notes covering the significant procedures, functions and libraries not yet mentioned. Most procedures and functions called in MacCAD are basic PASCAL commands. Some are characteristic only of the Macintosh and are located within the computers ROM. Others are included in the library called 'SANE', the Standard Apple Numeric Environment. Procedures and functions that have not yet been discussed in earlier sections are in the 'CAD NumberCrunch' module. The remaining are from the Programmer's Extender Volumes 1 and 2 libraries and modules. The SANE and Programmer's Extender libraries will be briefly discussed. The routines in the 'CAD NumberCrunch' module will be discussed in depth since they were written by the author especially for MacCAD.

B. SANE LIBRARY.

The Standard Apple Numeric Environment was established by Apple in an effort to standardize operations concerning the manipulation of numbers from one type to another and to and from strings. It also offers other functions such as exponential capabilities which are not usually available as basic PASCAL commands.

1. Number conversions

Extended numbers can be converted to integers, long integers, real, double precision, extended and decimal types as well as to a string. A variety of rounding procedures are also available using the variable 'RoundDir' to establish the desired rounding direction criteria.

2. Arithmetic functions

Such functions not normally available in PASCAL, supplied by the SANE library include Log base 2, and Ln and the ability to raise a base number to the power of an integer or a real number. The Tangent function is also available.

3. Financial and other miscellaneous functions

Although not used by MacCAD, financial annuity and compounding functions are included in SANE. There are functions concerning the class and sign of input numbers and others set or get the rounding direction as well as the rounding precision. Many other routines are available in the SANE library that are not applicable to MacCAD.

C. PROGRAMMER'S EXTENDER.

MacCAD was developed under the 'LightSpeed Pascal' environment. This is the compiler/linker/editor application that was used to write MacCAD. 'Programmer's Extender Volumes 1 and 2' are commercially available packages of libraries and subroutines written to be used in the 'LightSpeed Pascal' environment. Volume 1

offers functions and procedures that assist in the setting up and control of menus, windows, dialog/alert boxes, controls and other miscellaneous routines. Volume 2 contains routines for printing, window stacking, list controlling, popup menus, file input/output and the manipulation of bitmaps and pictures.

D. NUMBERCRUNCH ROUTINES.

This module started out as having some simple routines used to convert extended and integer type numbers to and from text strings. It grew into a library of routines that are shared by more than one module, such as handling number manipulation, alert/dialog box control and polynomial arithmetic functions.

1. WriteIt procedure

This is used to draw a number at some pixel location using the Quickdraw routine 'WriteDraw'. The number to be drawn and a criteria number, usually the same number, are entered as parameters. According to the size of the criteria number, the pen position will be moved by the appropriate number of pixels to the left or right so the drawn number will occupy the same location, regardless of the size of the number. Very large and very small numbers will be written in scientific notation. Numbers in between will have a varying number of digits displayed in order to maintain the same basic size in pixels.

2. FindSep and FindRealSep functions

These functions are used in calculating the number of divisions that are to be displayed on the axis of a plot. 'FindSep' is

used for integers, such as the magnitude axis of the bode plot. The max and min values are input as integers along with the starting number of divisions called 'initscale'. The function checks if 'initscale' can evenly be divided into the range between the max and min values input. If not, 'initscale' is alternately increased and decreased in integer steps until an even division is found. When it is found, the function returns it's value. 'FindRealSep' is used to find 'nice' divisions between extended numbers. The range as an extended number and the integer 'initscale' are entered. The range is successively multiplied by 10 until the resulting range is greater than 100. It is then changed into an integer and 'FindSep' is called to calculate the number of divisions. 'FindRealSep' then divides the resulting step size by the same number the range was multiplied by and returns it.

3. FrameDataError procedure

This is called whenever a dialog box is used and a parameter input by the user has been checked and found to be in error. 'FrameDataError' is called with a boolean expression, 'firsterr' signifying the first error found in the dialog box, and an integer signifying the dialog item of the data box. If 'firsterr' is true, this is the first error detected. In this case, an alert box is displayed instructing the user to check the values he has input. The box with the incorrect data in it is then outlined and that box is selected so the user can make the first correction without having to move with the mouse or tab key. If it is not the first error, only the box is outlined.

4. ClearAllWindows procedure

This is used before displaying dialog boxes in order to clear any plot windows that may be presently displayed. It checks if the front window exists, the window is hidden. This is repeated until all displayed windows are hidden.

5. FrameCircle procedure

This is like the 'FrameOval' Quickdraw routine except it draws circles. The circle's center coordinates and the desired radius, in pixels is entered and the data is converted to call 'FrameOval'.

6. Str2Real and Str2Int procedure

A string of text of type str255 is input. Each character in the string is checked to see if it makes a valid integer or extended number. 'pointused' is a boolean expression used to keep track of the use of a decimal point for the extended numbers. If the string has been determined to be correct, it is converted to real number with the SANE function 'Str2Num'. In the case of 'Str2Int', the number is then changed into an integer. For both procedures, if the string is in the proper format the boolean flag 'valid' is returned as true and 'realout' or 'intout' returns the value.

7. Int2Str and Real2Str functions

These functions change an integer or extended number into a string of text. The variable 'typedecform' is used to set the number style and the number of digits to display. The procedure 'Num2Str' is then called to make the change. The function is then returned as the resulting string.

8. SetDData procedure

Since data displayed in dialog boxes is always in text format, 'SetDData' is used to identify the data box of the desired dialog box and enter the desired text. The dialog pointer, dialog item and desired text are inputs for this procedure.

9. GetDData procedure

This is the same as 'SetDData' in reverse. The desired dialog box and it's item are identified and the text presently residing in that dialog item is returned as the str255 variable 'data'. This procedure is used when checking for proper number format of the dialog parameters input by the user.

10. GetCheckReal and GetCheckInt function and procedure

These functions call 'GetDData' and 'Str2Real' or 'Str2Int' described above, in order to get the parameter input by the user and check if it is in the proper format for extended or integer numbers. The same boolean 'valid' is returned along with the number, if 'valid'.

11. Basic1Alert and Basic2Alert procedures

These procedures display an alert box. Either one or two strings of text are input along with an integer from 0 to 3 to identify the type of alert to use. The types are Normal, Note, Caution and Stop. They are used most often to tell the user that the operation he has just selected is not appropriate for some reason. For example he may want to redraw a plot. It a plot has not yet been drawn, he will be notified by one of these procedures.

12. PolySum function

Polynomials in the coefficient form are often added or subtracted when calculating equivalent transfer functions. This is done by the 'PolySum' function. The two polynomials that the operation is to be done on are input along with the boolean 'addit' which is true if they are to be added and false if they are to be subtracted. The function sets the output polynomial's degree to be equal to the highest degree of the two polynomials input and adds their corresponding coefficients. The function is returned as the resulting polynomial.

13. PolyMult function

This function multiplies two input polynomials. It is returned as a boolean expression that is true if the sum of the two input polynomials' degrees are less than 20. This ensures that the resulting polynomial will fit within the 'polycoef' variable limits. Two nested 'for' loops do the multiplication and enter the product into the output polynomial.

14. FactToCoef function

A polynomial in factored format is entered and it is returned in coefficient format. This is done by forming a first order polynomial from each factor. The first order polynomials are successively multiplied together using 'PolyMult' from inside a 'while' loop until all factors have been used. The output polynomial's gain and order are set to those of the input and it is returned by the function.

15. PolyNorm procedure

This normalizes an input 'polycoef' by dividing all the coefficients by the coefficient of the order equal to the polynomial's degree. The gain constant is then multiplied by the same number. This makes the highest 's' term's coefficient equal to 1.

16. Log function

The log of the input number is calculated using the 'e' and natural log relationship. This is used primarily in the Bode plot calculations since the output is a semi-log plot. If the number 0 is entered, the output is returned as 1e-20. The function returns the calculated value of the Log of the input.

17. Ten2 function

This is used to convert back from the Log function mentioned above. It uses the 'XpwrY' function to calculate '10' raised to the power of the input number. The function 'Ten2' returns this number.

18. FindPhase function

This function effectively returns the phase information of a conversion from rectangular to polar coordinates. The input variable is of type 'complex', containing a real and imaginary element. From the signs of the elements, the quadrant of the number is identified and the result of the arctan function is adjusted accordingly. 'FindPhase' returns the angle in radians.

19. EvalGeq function

This function evaluates a transfer function in the 's' domain and returns it's magnitude and phase as a function of the frequency

that is input as 'freq'. The function returns the magnitude and the phase is returned as an output variable. The values are calculated for both the numerator and the denominator. The resulting magnitudes are divided and the phases subtracted to give the final values. For each polynomial, a counter is used starting at 1, increasing to the polynomial's degree. A 'case' statement and the 'mod' operator are used to determine if when 's' is replaced with j * 'freq', will the result be real or imaginary and what will be the sign after 's' is raised to it's appropriate power. The values are added in a variable of type 'complex'. The resulting magnitude is determined from the root of the sum of the squares of the complex elements and the phase is determined from the 'FindPhase' function.

20. Pole2Rect procedure

This procedure is called when calculating point positions for the Nyquist plot. It converts from polar to rectangular form with the output coordinates being in intergers for easy conversion to pixels. If the magnitude of either coordinate is larger than 500, which would put the point well off the screen, it is assigned the value of 500 in order to prevent floating point errors during later calculations with the point values.

21. DoQuad procedure

This solves for the roots of a 2nd order polynomial using the quadratic equation. It is called by the 'RootFinder' procedure which is described in the section covering the 'Root Finder' tool. The values for 'b' and 'c' are input according to the equation $x^2 + bx + c$,

the roots are solved. The roots are output in two variables of type 'complex'.

XI. CONCLUSIONS

A. SUMMARY OF PROGRAM

MacCAD offers a user friendly environment for the engineering student to design and analyze control systems ranging from simple single block transfer functions to complicated multiple loop systems consisting of several blocks. All of the usual graphic tools are available with the Bode, Nyquist, Root Locus and Time Response plots. The program offers additional options that are not available on other CAD programs. These include the ability to:

1. Overlap any number of plots.
2. Select either linear or logarithmic calculation point intervals.
3. To add automatically sized labels.
4. To print vertically, horizontally or with 50% reduction.
5. To enter or view any transfer function in either factored or coefficient form.
6. To add, change, delete, simplify or expand blocks easily.
7. To view several plots simultaneously.

These capabilities along with the standard Macintosh user friendly, mouse and menu driven operation lets the user spend more time designing and analyzing his system rather than trying to learn how to use a new computer program.

B. POTENTIAL FOR IMPROVEMENT

There are some options that could still be added to MacCAD. They include:

1. A two parameter Root Locus capability that lets the user input the polynomial coefficients as functions of two parameters and then drawing a family of curves in the form of a grid.

2. Saving data points calculated by the Nyquist plot to a text file for later viewing or use by a word processing application or desk accessory. The frequency, magnitude and phase for each point calculated could be entered in a file designated by the user as each point is being calculated.

3. The ability to select a gain value for the Root Locus and have all the roots at that gain displayed.

4. An on line help capability would be useful to the beginning user for quick reference while running the program.

These improvements would not be complicated to implement. With the 'project' building structure of 'Lightspeed Pascal' and the Macintosh's menu driven format, adding the additional modules with the changes would be very easy.

MacCAD is the first user friendly full function CAD program available that is both simple enough for the beginning student to easily use as well as powerful and flexible enough to benefit the experienced system designer.

APPENDIX

SOURCE CODE

The appendix contains the source code of the modules of MacCAD. Only the modules in the LightSpeed Pascal project written by the author are listed here. Standard Apple Macintosh libraries and the modules from the Programmer's Extender volumes are not included. The modules are not listed in the inverted reference order as they would appear in the project. A disk is available from Dr. Thaler that contains the MacCAD source code, the MacCAD resource file and the MacCAD application.

```
{**************************************************}
{                                                  }
{                    MacCAD                        }
{                                                  }
{                  version 2.0                     }
{                  02 Dec 1987                     }
{                                                  }
{                    author                        }
{                Kenneth S. MacDonald              }
{                                                  }
{**************************************************}
{                                                  }
{   system:    Apple Macintosh SE                  }
{   Language:  LightSpeed Pascal version 1.11      }
{   Libraries: Programmer's Extender vol 1 & 2 version 3.05 }
{   Resource:  MacCAD.rsrc                         }
{                                                  }
{        developed under System 4.2 and Finder 6.0 }
{                                                  }
{**************************************************}
{                                                  }
{      MacCAD is a computer aided design program designed for the }
{   graduate level student in Controls or Electrical Engineering. It }
{   offers a flexible block daigram manipulator as well as the standard }
{   graphic design tools, Bode, Nyquist, Root Locus and Time response }
{   plots. It can handle up to a 10th order 's' domain equivalent }
{   transfer function and offers the capability of overlapping plots and }
{   user defined plot calculation and display parameters. It has been }
{   checked to run on both 'old' and 'new' ROMs and requires at least }
{   512 K of RAM.                                  }
{                                                  }
{**************************************************}
program Main;
  uses
    XTTypeDefs, Extender1, CADSetUp, CADGlobals, AllClose, WindowTile, DoMenu;
  var
    openOrPrint, nofiles : integer;
    inforecord : appFile;
begin
  XTendInit;
  SetUp;
  CountAppFiles(openOrPrint, nofiles);
  if nofiles > 0 then
    OpenFile(false);    { open file}
  repeat
    repeat
      SystemTask;
      MaintainCursor(arrow, iBeam, arrow);
    until XTGetNextEvent(EveryEvent, event);
    HandleEvent(event, whatHappened);
    HandleMenu;
  until ExitRequest(whatHappened);
  CloseAll;
end.
```

```
unit CADGlobals;
Interface
 uses
  XTTypeDefs;
 const
{-----------------------RESOURCE   HANDLING------------------- }
   toolsmenuno = 200;   { menu id numbers }
   blocksmenuno = 300;
   windowmenuno = 400;
   aboutalertID = 300;   { resource id no for about mac cad alert }
   basic2id = 20240;  { multipurpse alert with editable text }
   basic1id = 12270;    { for single text alert boxes      }
   displaygrpid = 8246;    { displays blocks in group }
   fbackId = 32470;      { what type of feedback }
   blkorgrpid = 23853;     { display group or block }
   simpformid = 23791;   { ask simplification type}
   getstringid = 31883;    { ask for new block name }
   bodeselID = 24703;      { gets bode plot data   }
   rootid = 23781;       { ask for poly order to solve}
   polyfactid = 13532;     { displays factored roots }
   nyquistid = 1155;     { gets nyquist plot data }
   calcpointid = 25487;    { displays points during calculation }
   labeldid = 5111;      { gets label text strings }
   savelabelaID = 14805;  { alert to save new label}
   nyqalertid = 13753;     { alert for nyquist select }
   rlocusalertid = 20188;    { alert for root locus sel}
   rlocusid = 5883;      { dialog for root locus data}
{-------------------------BLOCK   HANDLING--------------------}
   maxbks = 5;
   maxorder = 20;
{------------------------------------------------------------- }
 type
{--------------------    BLOCK  DATA  STRUCTURES  ----------------  }
   complex = record  { complex number, real and imaginary parts}
      justreal : boolean;
      realpart : extended;
      imagpart : extended;
     end;
   polyfact = record        { factored polynomial }
      degree : integer;
      gain : extended;
      fact : array[1..20] of  complex;
     end;
   polycoef = record        { polynomial in coefficient form }
      degree : integer;
      gain : extended;
      coef : array[1..20] of extended;
     end;
   bksHdl = ^bksPtr;
   bksPtr = ^block;
   grpPtr = ^group;
   grpHdl = ^grpPtr;
   group = record
```

```
      ownHdl : grpHdl;
      maingrp : boolean;   {   is it the system group   }
      masterblock : bksHdl;   {   owner of the group   }
      fwdbks : integer;
      backbks : integer;
      bksused : array[1..5] of bksHdl;
      posFback : boolean;
    end;
  block = record          {   for each block in system   }
      title : string[255];
      used : boolean;
      changed : boolean;
      num : polycoef;
      den : polycoef;
      factored : boolean;
      forward : boolean;
      simplified : boolean;     {   made up from other blocks   }
      simpform : 1..4;
      subgrp : grpHdl;
      fromgrpHdl : grpHdl;    {   handle to group it is from   }
    end;
{-------------------PLOT  DATA  STRUCTURES----------------------}
  plotdata = record          {   data for bode and nyquist plots }
      minfreq : integer;
      maxfreq : integer;
      minmag : integer;
      maxmag : integer;
      layer : integer;
      doit : boolean;
    end;
{------------------------------------------------------------- }
 var
{----------------------------     MENU  HANDLEING  ----------------  }
  applemenu, fileMenu, editMenu, toolsmenu, blocksmenu, windowmenu : MenuHandle;
{----------------------------     CURSOR  HANDLEING  ----------------  }
  iBeam, cross, plus, watch : Cursor;
{----------------------------     EVENT  HANDLEING  ----------------  }
  event : EventRecord;   {   these types are in XTTypeDefs }
  whatHappened : EventStuff;
{----------------------------     WINDOW  HANDLEING  ---------------  }
  bodeR, rootR, nyqR, timeR : WindowRecord;
  bodePtr, rootPtr, nyqPtr, timePtr : WindowPtr;
  plotclipH : RgnHandle;                  {   for Nyquist plot clip region }
  radius : integer;                       {   nyquist plot radius   }
  firstnyquistrun : boolean;              {   ensure nyq clip rgn declared }
                                          {   only one time.              }
{----------------------------     RESOURCE  HANDLEING  ---------------  }
  fRefNum : integer;
{----------------------------     DIALOG  HANDLEING  ----------------  }
  Title, NumOrder, DenOrder, Path, Factored : str255;
        {   text input to dialog boxes   }
  itemType : integer;     {   used to get and change text from d-boxes }
  itemNum : integer;
```

```
    itemHndl : handle;
    DP : DialogPtr;
    displayrect : rect;
    m0, m1, m2, m3 : str255;
    ignore : integer;          { for alert boxes }
    saveit : boolean;          {  if cancel not hit on D-box  }
{----------------------------    BLOCK  HANDLEING  ----------------  }
    sysgroupH : grpHdl;
    sysblockH : bksHdl;
    unusedblock : block;
    unusedgrpPtr : grpPtr;
    unusedgrpHdl : grpHdl;
    noblock : block;
    unityblock : block;
    editnewblock : boolean;    {  inputting new blocks and root finder  }
{----------------------------    PLOT  HANDLEING  ----------------  }
    bodedata : plotdata;
    nyquistdata : record
        minfreq : extended;
        maxfreq : extended;
        minmag : integer;
        maxmag : integer;
        layer : integer;
        linear : boolean;
        pointstoplot : integer;
        doit : boolean;
      end;
    timedata : record
        inputtype : integer;
        maxtime : extended;
        amp : extended;
        impamp : extended;
        autoimpamp : boolean;
        zerobottom : boolean;
        freq : extended;
        slope : extended;
        dcoff : extended;
        maxy : extended;
        layer : integer;
        doit : boolean;
      end;
    RLocusdata : record
        mingain : extended;
        maxgain : extended;
        points : integer;
        xmin : extended;
        xmax : extended;
        ymin : extended;
        ymax : extended;
        linear : boolean;          {  true => linear, false => logrithmic }
        layer : integer;
        simptype : boolean;        {  true => Geq, false => closed loop  }
        doit : boolean;
```

```
    end;
  savereply : SFReply;    { for saving data to file }
implementation
end.
```

```
unit CADSetUp;

interface

 uses
  XTTypeDefs, Extender1, Extend2Stuff, CADGlobals, SANE;

 procedure InitBks;

 procedure SetUp;

implementation

{  ------------------------InitBks  procedure----------------------}
 procedure InitBks;
  var
   counter : integer;
 begin
  sysblockH := BksHdl(NewHandle(Sizeof(block)));   { define sysblockH }

  sysgroupH := grpHdl(NewHandle(Sizeof(group)));   { define sysgroupH }
  with sysgroupH^^ do
   begin
    ownHdl := sysgroupH;
    fwdbks := 0;
    backbks := 0;
    maingrp := true;
    masterblock := sysblockH;
    posFback := false;
   end;
  noblock.fromgrpHdl := sysgroupH;
  for counter := 1 to maxbks do
   begin
    sysgroupH^^.bksused[counter] := BKSHDL(NewHandle(Sizeof(BLOCK)));
    sysgroupH^^.bksused[counter]^^ := noblock;
   end;
  with sysblockH^^ do
   begin
    subgrp := sysgroupH;
    title := 'Main System';
    factored := true;
    forward := true;
    used := true;
    changed := false;
    simplified := true;
   end;
 end;
{   ------------------------------------------------------------}
 procedure SetUp;

{---------------------------  SetUpWindows  ----------------  }
{ Sets up windows used for tools menu.                 }
```

```
  procedure SetUpWindows;
    const
      top = 38;
      left = 0;
      bottom = 250;
      right = 400;
      moverect = 38;
    var
      temprect : rect;
  begin
    SetRect(temprect, left, top, right, bottom);
    bodePtr := CreateWindow(bodeR, temprect, 'Bode Plot', 8, false, true, true, true, true);
    OffsetRect(temprect, moverect, moverect);
    rootPtr := CreateWindow(rootR, temprect, 'Root Locus', 8, false, true, true, true, true);
    OffsetRect(temprect, moverect, moverect);
    nyqPtr := CreateWindow(nyqR, temprect, 'Nyquist Plot', 8, false, true, true, true, true);
    OffsetRect(temprect, moverect, moverect);
    timePtr := CreateWindow(timeR, temprect, 'Time Response', 8, false, true, true, true, true);
    TextFace([bold]);
  end;      { SetUpWindows }

{----------------------------      SetUpMenues  ----------------  }
  procedure SetUpMenues;
  begin
    m0 := '';
    m1 := '';
    m2 := '';
    m3 := '';
    StdMenus(applemenu, fileMenu, editMenu);
    DeleteMenu(fileid);
    DeleteMenu(editid);
    filemenu := BuildMenu(fileid, 'File', 'New;Open/O;Save;Save As;Print/P;Quit/Q');
    editmenu := BuildMenu(editid, 'Edit', 'Cut;Copy;Paste;Clear');
    blocksmenu := BuildMenu(blocksmenuno, 'Blocks', 'Change/E;Add New Block/A;Simplify To
      Block/S;Delete Block/D;Expand To Group/G');
    toolsmenu := BuildMenu(toolsmenuno, 'Tools', 'Bode Plot/B;Root Locus/R;Nyquist Plot/N;Time
      Response/T;Root Finder/F;Add Label/L');
    windowmenu := BuildMenu(windowmenuno, 'Windows', 'Full Tile/W;Vert Tile/V;Horiz
      Tile/H;Stack/X;Show Plots/Z;Move Back/M;Close Front/C');
    SetMenuItem(applemenu, 1, 'About MacCAD');
    EnableAllItems(applemenu, true);
    EnableAllItems(filemenu, true);
    EnableAllItems(editmenu, true);
    EnableAllItems(windowmenu, true);
  end;    { SetUpMenues }
{  ----------------------InitUnityBlock   procedure----------------------}
  procedure InitUnityBlock;
    var
      counter : integer;
  begin
    with noblock do
      begin
```

```
      used := false;
      title := 'no block';
      changed := false;
      simplified := false;
    end;
  with unityblock do
    begin
      factored := false;
      used := true;
      with num do
        begin
          gain := 1.0;
          degree := 0;
          coef[1] := 1.0;
          for counter := 2 to 20 do
            coef[counter] := 0;
        end;
      with den do
        begin
          gain := 1.0;
          degree := 0;
          coef[1] := 1.0;
          for counter := 2 to 20 do
            coef[counter] := 0;
        end;
    end;
end;

{  ----------------------InitPlotData   procedure-----------------------}

procedure InitPlotData;
begin
  with nyquistdata do
    begin
      minfreq := 0.01;
      maxfreq := 1000;
      minmag := 0;
      maxmag := 5;
      linear := false;
      layer := 0;
      pointstoplot := 200;
      doit := true;
    end;
  firstnyquistrun := true;  { so the clip region will be declared only once }
  with bodedata do
    begin
      minfreq := -2;
      maxfreq := 3;
      minmag := -40;
      maxmag := 40;
      layer := 0;
      doit := true;
    end;
```

```
    with timedata do
      begin
        inputtype := 1;
        maxtime := 10;
        amp := 1;
        impamp := 1000 / maxtime;
        autoimpamp := true;
        zerobottom := true;
        freq := 1;
        slope := 1;
        dcoff := 0;
        maxy := 1.25;
        doit := true;
        layer := 0;
      end;
    with RLocusdata do
      begin
        mingain := 0.1;
        maxgain := 10;
        xmin := -10;
        xmax := 2;
        ymin := -10;
        ymax := 10;
        points := 25;
        linear := true;
        layer := 0;
        simptype := true;
        doit := true;
      end;{ with RLocus}
  end;

{   ----------------------SetUp   procedure----------------------}

  begin
    HideAll;
    fRefNum := OpenResFile('MacCAD.Rsrc');
    SetUpMenues;
    InitUnityBlock;
    InitBks;
    InitPlotData;
    savereply.good := false;
    FetchCursors(iBeam, cross, plus, watch);
    ignore := Alert(aboutalertID, nil);
    SetUpWindows;
  end;



end.
```

```
unit DoMenu;

interface

 uses
   XTTypeDefs, Extender1, CADGlobals, Nyquist, SANE, Time, NumberCrunch, DoBlockMenu, simpgroup,
       bode, RFinder, AddLabel, RootLocus, CADSetUp, WindowTile, PrintWindow;

 procedure HandleMenu;

implementation

 procedure HandleMenu;

{-------------------------------HandleAppleMenu---------------------}
 Calls alert box for About MacCad                    }
   procedure HandleAppleMenu;
   begin
     with whatHappened do
       case itemNum of
         1 :
           begin
             ignore := Alert(aboutalertID, nil);
           end;
         otherwise
           ;
       end;
   end; {  HandleAppleMenu  }
{-------------------------------HandleFileMenu---------------------}
 Calls procedures needed by File menu selections          }
   procedure HandleFileMenu;
   begin
     with whatHappened do
       case itemNum of
         1 :    { new }
           begin
             ignore := CautionAlert(5901, nil);
             if ignore = 2 then
               begin
                 savereply.good := false;
                 InitBks;
                 DoAddBlock;
               end;
           end;
         2 :  { Open }
           begin
             OpenFile(false);
           end;

         3 :    { save }
           begin
 avereply.good := true;}
             SaveFile;
```

```
        end;
      4 :    { save as }
        begin
          savereply.good := false;
          SaveFile;
        end;
      5 :    { print}
        DoPrintMenu;
      otherwise
        ;
    end { case }
  end;
{--------------------------------HandleBlocksMenu---------------------}
{ Calls procedures needed by Block menu selections. Calls DoBlockMenu procedures}

  procedure HandleBlocksMenu;
  begin
    ClearAllWindows;
    with whatHappened do
      case ItemNum of
        1 :        { Change }
          begin
            ChangeBlock;
          end;
        2 :  { Add Block }
          DoAddBlock;
        3 :  { Simplify }
          begin
            if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
              DoSimplifyGroup
            else
              Basic1Alert('There are no blocks in the system.', 1);
          end;
        4 : { Delete}
          begin
            DeleteBlock;
          end;
        5 :{ Expand }
          ExpandBlock;
        otherwise
          ;
      end; { ItemNum case }
  end;    { HandleBlocksMenu }
{-----------------------------HandleWindowMenu---------------------}

  procedure HandleWindowMenu;
  begin
    with whatHappened do
      DoWindowMenu(itemnum);
  end;    { HandleWindowMenu }

{--------------------------------HandleToolsMenu---------------------}
  procedure HandleToolsMenu;
```

```
  begin
    with whatHappened do
      case ItemNum of
        1 :         { Bode Plot }
          DoBodeMenu;
        2 :         { Root Locus }
          begin
            DoRLocusMenu;
          end;
        3 :       { Nyquist }
          DoNyquistMenu;
        4 : { Time Response }
          begin
            DoTimeMenu;
          end;
        5 :    { Root Finder }
          begin
            ClearAllWindows;
            DoRFinderMenu;
          end;
        6 :{ Label}
          DoLabelMenu;
        otherwise
          ;
      end; { ItemNum case }
  end;    { HandleToolsMenu }
-------------------------------------------------------------------------}
------------------HandleMenu   procedure-----------------------------}
  var
    temperr : OSerr;
begin
  with whatHappened do
    case MenuNum of
      appleid :           { apple menu selected }
        HandleAppleMenu;
      toolsmenuno :      { Tools menu selected }
        HandleToolsMenu;
      blocksmenuno : { Blocks Menu selected }
        HandleBlocksMenu;
      fileID :
        HandleFileMenu;
      editID :
        temperr := DoEditMenu(whatHappened);
      windowmenuno :
        HandleWindowMenu;
      otherwise
        ;
    end;    { with MenuNum }
end;

nd.
```

```
unit NumberCrunch;
interface
 uses
  XTTypeDefs, Extender1, CADGlobals, sane;
 procedure WriteIt (numout : extended;
             criteria : extended);
 function FindSep (max, min : longint;
             initscale : integer) : integer;
 function FindRealSep (var max : extended;
             initscale : integer) : extended;
 procedure FrameDataError (var firsterr : boolean;
             item : integer);
 procedure ClearAllWindows;
 procedure FrameCircle (x, y : integer;
             rad : extended);
 procedure Str2Real (stringin : str255;
             var realout : extended;
             var valid : boolean);
 procedure Str2Int (stringin : string;
             var Intout : integer;
             var valid : boolean);
 procedure Pole2Rect (mag, phase : extended;
             var x : integer;
             var y : integer);
 function Log (input : extended) : extended;
 function EvalGeq (blocktouse : block;
             freq : extended;
             var phase : extended) : extended;
 function Int2Str (intin : integer) : string;
 function Real2Str (realin : extended;
             accurate : boolean) : string;
 procedure SetDData (DP : DialogPtr;
             itemno : integer;
             data : str255);
 procedure GetDData (DP : DialogPtr;
             itemno : integer;
             var data : str255);
 function GetCheckReal (DP : DialogPtr;
             Itemno : integer;
             var realout : extended) : boolean;
 procedure GetCheckInt (DP : DialogPtr;
             Itemno : integer;
             var Intout : integer;
             var valid : boolean);
 procedure DoQuad (b, c : extended;
             var comp1, comp2 : complex);
 function Ten2 (exponent : extended) : extended;
 procedure RootFinder (polycin : polycoef;
             var polyfout : polyfact;
             accurate : boolean);
 procedure PolyNorm (var polyin : polycoef);
 function PolyMult (poly1 : polycoef;
             poly2 : polycoef;
```

```
            var polyout : polycoef) : boolean;
 function PolySum (poly1 : polycoef;
            Addit : boolean;
            poly2 : polycoef) : polycoef;
 function FactToCoef (polyin : polyfact) : polycoef;
 procedure Basic1Alert (textout : str255;
            alerttype : integer);
 procedure Basic2Alert (text1, text2 : str255;
            alerttype : integer);
implementation
{-------------------WriteIt   procedure------------------}
 procedure WriteIt;{(numout : extended;criteria : extended );}
 begin
  if (abs(criteria) < 0.001) or (abs(criteria) > 1000) then
    WriteDraw(numout)
  else if abs(criteria) < 0.01 then
    WriteDraw(numout : 4 : 4)
  else if abs(criteria) < 1 then
    WriteDraw(numout : 4 : 3)
  else
    WriteDraw(numout : 8 : 1);
 end;
----------------------------FindSep   function----------------------}
 function FindSep;{(max, min:longint, initscale : integer) : integer;}
  var
    scale, interval : integer;
 begin
  scale := initscale;
  interval := 0;
  while (max - min) mod scale <> 0 do
    begin
      interval := interval + 1;
      if scale <= initscale then
        scale := scale + interval
      else
        scale := scale - interval;
    end;
  FindSep := (max - min) div scale;
 end;
-------------------------FindRealSep   function----------------------}
 function FindRealSep;{(var max : extended;initscale : integer) : extended;}
  var
    newmax : extended;
    maxint : longint;
    multiplier : extended;
 begin
  multiplier := 1;
  repeat
    multiplier := multiplier * 10.0;
    newmax := multiplier * max;
  until newmax > 100.000001;
  maxint := Num2Longint(newmax);
  max := maxint / multiplier;
```

```
    FindRealSep := FindSep(maxint, 0, initscale) / multiplier;
  end;
{-----------------------FrameDataError   procedure-----------------}
  procedure FrameDataError; {(firsterr : boolean;item : integer)}
  begin
    if firsterr then
      begin
        hidewindow(DP);
        Basic1Alert('Please ensure you are inputting appropriate values.', 1);  { show error alert  }
        showwindow(dp);
        SelIText(DP, item, 0, 255);
        firsterr := false;
      end;
    FrameDItem(DP, item);
  end;
{-------------------------ClearAllWindows   procedure--------------------}
  procedure ClearAllWindows;
    var
      windowtoclear : WindowPtr;
  begin
    repeat
      windowtoclear := FrontWindow;
      if windowtoclear <> nil then
        HideWindow(windowtoclear);
    until windowtoclear = nil;
  end;
{-------------------------FrameCircle   procedure--------------------}
  procedure FrameCircle;  { (x, y:integer,rad : extended);}
    var
      radint : integer;
  begin
    radint := Num2Integer(rad);
    FrameOval(y - radint, x - radint, y + radint, x + radint);
  end;
{--------------------------------Str2Real-----------------------------}
{ checks each character in text to ensure that is real number. "Valid" is the  }
{ boolean error checking flag. If it is good, the text is changed to a real num.  }
  procedure Str2Real;
    var
      firstchar, otherchar, onlychar : set of char;
      tempchar : char;
      counter : integer;
      pointused : boolean;
  begin
    pointused := false;     { flag showing if decimal point has been used. }
    valid := true;          { number is good until proven otherwise  }
    firstchar := ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '.'];{ valid first characters }
    onlychar := ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0'];
    otherchar := ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '.']; { valid other characters }
    if length(stringin) = 1 then
      begin
        tempchar := copy(stringin, 1, 1);   { get first character    }
        if not (tempchar in onlychar) then
```

```
      valid := false;
    end
  else if length(stringin) > 1 then
    begin
      tempchar := copy(stringin, 1, 1);    { get first character    }
      if tempchar in firstchar then      { check for good first char  }
{ if first is good, check the rest up through the length of the text     }
        begin
          if tempchar = '.' then { check if first char was a decimal point }
            pointused := true;    { decimal point has been used.}
          for counter := 2 to length(stringin) do
            begin
              tempchar := copy(stringin, counter, 1);
              if (tempchar in otherchar) then { if other chars are good}
                begin
                  if (tempchar = '.') then
                    if pointused then
                      valid := false
                    else
                      pointused := true;
                end
              else
                valid := false;
            end;
        end
      else
        valid := false;    { if first char is not good }
    end
  else
    valid := false;
  if valid then
    realout := str2Num(stringin) { if good, make the change}
  else
    valid := false;
end;
{---------------------------------Str2Int-----------------------------------}
{checks each character in text to ensure that is an integer. "Valid" is the      }
{boolean error checking flag. If it is good, the text is changed to ainteger.  }
{see STr2Real for explaination of procedure code                  }
procedure Str2Int;
  var
    goodchar, firstchar : set of char;
    tempchar : char;
    counter : integer;
begin
  valid := true;
  firstchar := ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-'];
  goodchar := ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0'];
  if length(stringin) = 1 then
    begin
      tempchar := copy(stringin, 1, 1);
      if not (tempchar in goodchar) then
        valid := false;
```

```
    end
  else if length(stringin) > 1 then
    begin
      tempchar := copy(stringin, 1, 1);
      if not (tempchar in firstchar) then
        valid := false;
      if length(stringin) > 1 then
        for counter := 2 to length(stringin) do
          begin
            tempchar := copy(stringin, counter, 1);
            if not (tempchar in goodchar) then
              valid := false;
          end;
    end
  else
    valid := false;
  if valid then
    Intout := Num2Integer(str2Num(stringin))
  else
    Intout := 0;
end;
{---------------------------------Int2Str-----------------------------------}
{ changes an integer to a string for returning to dialog boxes              }
function Int2Str;
  var
    tempext : extended;
    typedecform : decform;
    tempstring : decstr;
begin
  typedecform.style := FixedDecimal;
  typedecform.digits := 0;
  tempext := intin;
  num2str(typedecform, tempext, tempstring);
  Int2Str := tempstring;
end;
{---------------------------------Real2Str----------------------------------}
{ changes a real no. to a string for return to dialog boxes                 }
function Real2Str;
  var
    typedecform : decform;
    tempstring : decstr;
begin
  typedecform.style := fixeddecimal;
  typedecform.digits := 15;
  num2str(typedecform, realin, tempstring);
  Real2Str := tempstring;
end;
{---------------------------------SetDData----------------------------------}
{ puts the input text to the editable dialog box location       }
procedure SetDData;
begin
  GetDItem(DP, itemno, itemType, itemhndl, displayrect);
  SetIText(itemhndl, data);
```

```
  end;
{--------------------------------GetDData---------------------------------}
{ returns the text string located at the item number in the dialog box indicated      }
  procedure GetDData;
  begin
    GetDItem(DP, itemno, itemType, itemhndl, displayrect);
    GetIText(itemhndl, data);
  end;
{-------------------------------GetCheckReal-------------------------------}
  function GetCheckReal;
    var
      tempstring : str255;
      valid : boolean;
  begin
    realout := 0;
    GetDData(DP, itemno, tempstring);
    Str2Real(tempstring, realout, valid);
    GetCheckReal := valid;
  end;
{-------------------------------GetCheckInt--------------------------------}
  procedure GetCheckInt;
    var
      tempstring : str255;
  begin
    GetDData(DP, itemno, tempstring);
    Str2Int(tempstring, intout, valid);
  end;
{-------------------------------Basic1Alert--------------------------------}
  procedure  Basic1Alert;{(textout:str255;alerttype:integer)}
  begin
    m3 := textout;
    ParamText(m0, m1, m2, m3);
    case alerttype of
      0 :
        ignore := Alert(basic1id, nil);
      1 :
        ignore := NoteAlert(basic1id, nil);
      2 :
        ignore := CautionAlert(basic1id, nil);
      3 :
        ignore := StopAlert(basic1id, nil);
      otherwise
        ;
    end; { case}
    m3 := '';
    ParamText(m0, m1, m2, m3);
  end;
{-------------------------------Basic2Alert--------------------------------}
  procedure  Basic2Alert;{(text1:str255;text2:str255;alerttype:integer)}
  begin
    m2 := text1;
    m3 := text2;
    ParamText(m0, m1, m2, m3);
```

```
    case alerttype of
      0 :
        ignore := Alert(basic2id, nil);
      1 :
        ignore := NoteAlert(basic2id, nil);
      2 :
        ignore := CautionAlert(basic2id, nil);
      3 :
        ignore := StopAlert(basic2id, nil);
      otherwise
        ;
    end; { case}
    m2 := '';
    m3 := '';
    ParamText(m0, m1, m2, m3);
  end;
{--------------------------------PolySum--------------------------------}
  function PolySum;
{ ( poly1 : polycoef; Addit : boolean; poly2 : polycoef ) : polycoef;}
    var
      plus, tempxa, tempxb, tempxc : extended;
      counter : integer;
      temppoly : polycoef;
  begin
    for counter := (poly1.degree + 2) to 20 do
      poly1.coef[counter] := 0;
    for counter := (poly2.degree + 2) to 20 do
      begin
        poly2.coef[counter] := 0;
        temppoly.coef[counter] := 0;
      end;
    if poly1.degree > poly2.degree then
      temppoly.degree := poly1.degree
    else
      temppoly.degree := poly2.degree;
    if addit then
      plus := 1.0
    else
      plus := -1.0;
    for counter := 1 to 20 do
      temppoly.coef[counter] := poly1.gain * poly1.coef[counter] + plus * poly2.gain * poly2.coef[counter];
    temppoly.gain := 1.0;
    PolySum := temppoly;
  end;
{--------------------------------PolyMult--------------------------------}
  function PolyMult;
    {    ( poly1 : polycoef; poly2 : polycoef; var polyout : polycoef    ) : boolean;}
    var
      x, y : integer;
      tempA, tempB, tempC : extended;
  begin
    if poly1.degree + poly2.degree < 20 then
      begin
```

```
    for x := 1 to 20 do
      polyout.coef[x] := 0;
    for x := 1 to poly1.degree + 1 do
      begin
        for y := 1 to poly2.degree + 1 do
          begin
            polyout.coef[x + y - 1] := poly1.coef[x] * poly2.coef[y] + polyout.coef[x + y - 1];
          end;
      end;
    PolyMult := true;
  end
 else
  PolyMult := false;
 polyout.degree := poly1.degree + poly2.degree;
 polyout.gain := poly1.gain * poly2.gain;
end;
{--------------------------------FacttoCoef--------------------------------}
function FactToCoef;
{ ( polyin : polyfact ) : polycoef; }
 var
  startpoly, nextpoly : polycoef;
  a, b : extended;
  counter : integer;
  onlyreal : boolean;
begin
 counter := 1;
 startpoly.gain := 1.0;
 startpoly.degree := 0;
 startpoly.coef[1] := 1;
 nextpoly.gain := 1;
 while counter <= polyin.degree do
  begin
    a := polyin.fact[counter].realpart;
    b := polyin.fact[counter].imagpart;
    onlyreal := polyin.fact[counter].justreal;
    if onlyreal then
      begin
        nextpoly.degree := 1;
        nextpoly.coef[2] := 1;
        nextpoly.coef[1] := a;
        counter := counter + 1;
      end
    else
      begin
        nextpoly.degree := 2;
        nextpoly.coef[3] := 1;
        nextpoly.coef[2] := 2 * a;
        nextpoly.coef[1] := sqr(a) + sqr(b);
        counter := counter + 2;
      end;
    if PolyMult(startpoly, nextpoly, startpoly) then
      ;
  end;
```

```
    startpoly.gain := polyin.gain;
    FactToCoef := startpoly;
  end;
{--------------------------------PolyNorm--------------------------------}
  procedure PolyNorm; {(var polyin : polycoef);}
{ this normalizes a polynomial by dividing all values by most significant }
{ coef.                                  }
    var
      normno : extended;
      counter, n : integer;
  begin
    n := polyin.degree;
    n := n + 1;    { ids most sig coef }
    normno := polyin.coef[n];
    polyin.gain := polyin.gain * normno;
    if normno <> 0 then
      for counter := 1 to n do
        begin
          polyin.coef[counter] := polyin.coef[counter] / normno;
        end;
  end;
{------------------------Log    procedure-------------------------}
  function Log; {(input : extended) : extended}
  begin
    if input = 0 then
      input := 1e-20;
    Log := 0.434294482 * ln(input);
  end;
{------------------------Ten2    function-------------------------}
  function Ten2; {(exponent : extended) : extended}
  begin
    Ten2 := XpwrY(10.0, exponent);
  end;
{------------------------FindPHase    function-------------------------}
{ input a complex value and the arctan, in radians is returned.        }
  function FindPhase (value : complex) : extended;
    const
      halfpi = 1.5707963;
    var
      theta, phase : extended;
  begin
    if value.realpart <> 0 then
      begin
        theta := abs(value.imagpart / value.realpart);
        if value.realpart > 0 then
          begin
            if value.imagpart >= 0 then
              phase := arctan(theta)
            else
              phase := arctan(-theta) + 4 * halfpi;
          end
        else
          begin
```

```
        if value.imagpart > 0 then
          phase := -arctan(theta) + 2 * halfpi
        else
          phase := arctan(theta) + 2 * halfpi;
      end;
    end
  else
    begin      { if the real part is zero, phase = +/- 1/2 pi }
      if value.imagpart > 0 then
        phase := halfpi
      else
        phase := 3 * halfpi;
    end;
  FindPhase := phase;

end;
-----------------------EvalGeq    function-------------------------}
function EvalGeq;{(blocktouse : block, freq : extended var phase:extended) : extended;}
  var
    totmag, newmag : extended;
    newphase, totphase : extended;
    nummag, denmag : complex;
    counter : integer;
begin
  nummag.realpart := 0;
  nummag.imagpart := 0;
  denmag.realpart := 0;
  denmag.imagpart := 0;
  with blocktouse do
    begin
      with num do
        begin
          for counter := 1 to degree + 1 do
            begin
              newmag := coef[counter] * Xpwrl(freq, counter - 1);
              case ((counter - 1) mod 4) of
                0 :              {s raised to 0, 4, 8... powers }
                  nummag.realpart := nummag.realpart + newmag;
                1 :              {s raised to 1,5,9... powers}
                  nummag.imagpart := nummag.imagpart + newmag;
                2 :              {s raised to 2,6,10... powers }
                  nummag.realpart := nummag.realpart - newmag;
                3 :              {s raised to 3,7,11... powers }
                  nummag.imagpart := nummag.imagpart - newmag;
              end;{case num }
            end;  {for counter }
        end; {with num}
      newphase := FindPhase(nummag);
      totmag := num.gain * sqrt(sqr(nummag.realpart) + sqr(nummag.imagpart));
      with den do
        begin
          for counter := 1 to degree + 1 do
            begin
```

```
            newmag := coef[counter] * Xpwrl(freq, counter - 1);
            case ((counter - 1) mod 4) of
              0 :           {s raised to 0, 4, 8... powers }
                denmag.realpart := denmag.realpart + newmag;
              1 :           {s raised to 1,5,9... powers}
                denmag.imagpart := denmag.imagpart + newmag;
              2 :           {s raised to 2,6,10... powers }
                denmag.realpart := denmag.realpart - newmag;
              3 :           {s raised to 3,7,11... powers }
                denmag.imagpart := denmag.imagpart - newmag;
            end;{case den }
          end;  {for counter }
        end; {with den }
      phase := newphase - FindPhase(denmag);
      EvalGeq := (totmag / (den.gain * sqrt(sqr(denmag.realpart) + sqr(denmag.imagpart))));
    end;  {with blocktouse}
  end;
{-------------------------Pole2Rect    procedure---------------------------}
  procedure Pole2Rect;  { mag,phase:extended; var x,y:integer)}
    var
      tempvalue : extended;
  begin
    x := 500;
    y := 500;
    tempvalue := mag * cos(phase);
    If abs(tempvalue) < 500 then
      x := Num2Integer(tempvalue);
    tempvalue := -mag * sin(phase);
    If abs(tempvalue) < 500 then
      y := Num2Integer(tempvalue);    { y is mult by -1 because of Mac screen layout }
  end;
{---------------------------------DoQuad---------------------------------}
  procedure DoQuad;         {( b , c : extended;var comp1, comp2 : complex)}
{ puts (s^2 +bs +c) into form (s+comp1)(s+comp2)}
    var
      radno, realno, four, two : extended;
  begin
    four := 4.00;
    two := 2.00;
    radno := b * b / four - c;
    realno := b / two;
    If radno >= 0 then
      begin
        comp1.justreal := true;
        comp2.justreal := true;
        comp1.realpart := realno + sqrt(radno);
        comp2.realpart := realno - sqrt(radno);
        comp1.imagpart := 0.0;
        comp2.imagpart := 0.0;
      end
    else
      begin
        comp1.justreal := false;
```

```
    comp2.justreal := false;
    comp1.realpart := realno;
    comp2.realpart := realno;
    comp1.imagpart := sqrt(-radno);
    comp2.imagpart := -sqrt(-radno);
  end;
end;
{--------------------------------RootFinder--------------------------------}
procedure RootFinder;  {(polycin : polycoef; var polyfout : polyfact;accurate:boolean);}
  RootFinder changes a coef poly to a fact poly, by findeing the roots }
  of the coef equation.                }
  const
    maxit = 5000;
  var
    a, b, c : array[1..23] of extended;
    P, Q, delP, delQ, denom, epsilon : extended;
    iteration, counter, n : integer;
    finished, firstit : boolean;
    multiplier : extended;
    qispos, qwaspos, pispos, pwaspos : boolean;
    inlimits : boolean;
begin
  pwaspos := true;
  qwaspos := true;
  if accurate then
    epsilon := 0.0000001
  else
    epsilon := 0.01;
  PolyNorm(polycin);
  n := polycin.degree;
  polyfout.gain := polycin.gain;
  polyfout.degree := n;
  while (polycin.coef[1] = 0.0) and (n > 0) do          {   if s=0 is a root   }
    begin
      with polyfout.fact[n] do                  {   load first root value }
        begin
          realpart := 0;
          imagpart := 0;
          justreal := true;
          finished := true;
        end;
      for counter := 1 to n + 1 do              {   shift the coef values }
        polycin.coef[counter] := polycin.coef[counter + 1];
      n := n - 1;                               {   decrease the 'order' }
    end;
  for counter := 3 to n + 3 do
    A[counter] := polycin.coef[n + 4 - counter];{   load poly coef into A array }
  for counter := 1 to 23 do
    B[counter] := 0.0;
  C := B;
  iteration := 1;
  finished := false;
  P := 0.0;
```

```
    if A[4] < 0 then
      P := -2;
    Q := 0.0;
    delP := 0;
    delQ := 0;
    multiplier := 1.0;

{   if poly is of order 2 or less }
    case n of
      0 :
        begin
          finished := true;   {   poly of order 0   }
        end;
      1 :
        with polyfout.fact[1] do
          begin
            realpart := a[4];
            imagpart := 0;
            justreal := true;
            finished := true;
          end;
      2 :
        begin {   poly is quadradic }
          DoQuad(A[4], A[5], polyfout.fact[1], polyfout.fact[2]);
          finished := true;
        end;
      otherwise
        ;
    end; {   case   }
{       BEGINNING OF BAIRSTOWS METHOD OF ITERATION      }
    while (not finished) and (iteration < maxit) do
      begin
        for counter := 3 to n + 3 do
          begin
            B[counter] := A[counter] - P * B[counter - 1] - Q * B[counter - 2];
            C[counter] := B[counter] - P * C[counter - 1] - Q * C[counter - 2];
          end;

        denom := C[n + 1] * C[n + 1] - (C[n + 2] - B[n + 2]) * C[n];
        if denom <> 0 then
          begin
            delP := (B[n + 2] * C[n + 1] - B[n + 3] * C[n]) * multiplier / denom;
            delQ := (C[n + 1] * B[n + 3] - (C[n + 2] - B[n + 2]) * B[n + 2]) * multiplier / denom;
            P := P + delP;
            Q := Q + delQ;
            if (abs(P) > epsilon) and (abs(Q) < epsilon) then          {   if Q is very small    }
              inlimits := (abs(delP / P) + abs(delQ)) < epsilon
            else if (abs(P) < epsilon) and (abs(Q) > epsilon) then     {   if P is small, just imag  }
              inlimits := (abs(delP) + abs(delQ / Q)) < epsilon        {   roots                     }
            else if (abs(P) > epsilon) and (abs(Q) > epsilon) then     {   both P & Q are large      }
              inlimits := ((abs(delP / P) + abs(delQ / Q)) < epsilon)
            else                                                       {   both are small            }
              inlimits := ((abs(delP) + abs(delQ)) < epsilon);
```

```
    If inlimits then                                    { limit requirements met }
      begin
        DoQuad(P, Q, polyfout.fact[n], polyfout.fact[n - 1]);
        n := n - 2;
        case n of
          0 :
            finished := true;
          1 :
            begin
              polyfout.fact[n].realpart := B[n + 3] / b[n + 2];
              polyfout.fact[n].imagpart := 0.0;
              polyfout.fact[n].justreal := true;
              finished := true;
            end;
          2 :
            begin
              DoQuad(B[n + 2], B[n + 3], polyfout.fact[n], polyfout.fact[n - 1]);
              finished := true;
            end;
          otherwise
            begin
              A := B;
              iteration := 1;
              multiplier := 1.0;
              p := 0;
              q := 0;
              pwaspos := true;
              qwaspos := true;
            end;
        end; {case}
      end
    else
      iteration := iteration + 1;
  end
else
  begin
    P := P + 1;
    Q := Q + 1;
    iteration := 1;
  end;
If delP > 0 then
  pispos := true
else
  pispos := false;
If delQ > 0 then
  qispos := true
else
  qispos := false;
If ((pwaspos = not pispos) and (q = 0) or (p = 0) and (qwaspos = not qispos) or (pwaspos = not
 pispos) and (qwaspos = not qispos)) and (iteration > 1) then
  multiplier := multiplier / 2;
pwaspos := pispos;
qwaspos := qispos;
```

```
    end;{  while }
   If iteration = maxit then
     Basic2Alert('Run Time Error.', ' Do not accept the data. It will be unreliable. Sorry!', 2);
 end;
{--------------------------------END-ALL--------------------------------}
end.
```

```
unit DoBlockMenu;
interface
 uses
   XTTypeDefs, Extender1, CADGlobals, NumberCrunch, sane, simpgroup;

 var
   fbackchanged : boolean;

 function LoadPolyCoefErrorCheck (DP : DialogPtr;
             var polyin : polycoef) : boolean;

 function LoadPolyFactErrorCheck (DP : DialogPtr;
             var polyin : polyfact) : boolean;

 procedure CheckPolyCoef (var polycin : polycoef);

 procedure LoadPolyCoef (textout : string;
             var polyin : polycoef);

 procedure LoadPolyFact (textout : string;
             var polyin : polyfact);

 procedure ShiftPolyc (var polycin : polycoef);

 procedure GetFactoredData (var tempblock : block);

 procedure OldCoefDataOut (var polyout : polycoef);

 procedure OldFactDataOut (polyin : polyfact);

 procedure GetCoefData (var tempblock : block);

 procedure FrameError (DP : DialogPtr;
             itemno : integer;
             alertid : integer;
             var flag : boolean);

 procedure PutBlockInGroup (var tempblock : block;
             var tempgroup : group);

 function AddBlockErrorCheck (DP : DialogPtr;
             var tempgroup : group;
             var tempblock : block) : boolean;

 procedure DisplayGroup (titleout : str255;
             var blockoutHdl : bksHdl;
             var groupout : grpHdl;
             var showit : boolean;
             exceptsimplified : boolean);


 procedure LoadAddBlockData (var blocktochange : block);
```

```
  procedure UpdateData (var groupchanged : group);

  procedure DoSimplifyGroup;

  procedure ChangeBlock;

  procedure DeleteBlock;

  procedure EditBlock (var grouptochange : group;
                       var blocktochange : block;
                       newblock : boolean);

  procedure DoAddBlock;

  procedure CheckBlockGone (blockgone : block);

  procedure ExpandBlock;

implementation
{--------------------------------AddBlock    Section-----------------------------}

{--------------------------------FrameError--------------------------------}
{   frames the box with the data that is in the incorrect format                }
  procedure FrameError;
  begin
    if (not flag) then          {   error alert has not yet been shown     }
      begin
        hidewindow(DP);
        ignore := StopAlert(alertid, nil);      {   show error alert     }
        showwindow(dp);
        SelIText(DP, itemno, 0, 255);
        flag := true;          {   set flag that error alert shown and there was error }
      end;
    FrameDItem(DP, itemno);
  end;
{---------------------------LoadPolyCoefErrorCheck---------------------------}
  function LoadPolyCoefErrorCheck;
    var
      counter, index : integer;
      showerrbox : boolean;
      realout : extended;
  begin
    m3 := 'Input error. Please check that real numbers are being entered.';
    ParamText(m0, m1, m2, m3);
    showerrbox := false;
    if getcheckreal(DP, 3, realout) then
      polyin.gain := realout
    else
      FrameError(DP, 3, basic1id, showerrbox);
    index := polyin.degree + 1;
    for counter := 4 to polyin.degree + 4 do
      begin
        if GetcheckReal(DP, counter, realout) then
```

```
      polyin.coef[index] := realout
    else
      FrameError(DP, counter, basic1id, showerrbox);
    index := index - 1;
  end;
 for counter := polyin.degree + 2 to 20 do
  polyin.coef[counter] := 0;
 loadpolycoeferrorcheck := not showerrbox;
end;

{------------------------------LoadPolyFactErrorCheck--------------------------------}

function LoadPolyFactErrorCheck;
                                  {(DP : DialogPtr; var polyin : polyfact)}
 var
  counter, index, realno, imagno, order : integer;
  showerrbox : boolean;
  realout : extended;
  textreal, textimag : str255;
begin
 order := 0;
 m3 := 'Input error. Please check that real numbers are being entered.';
 ParamText(m0, m1, m2, m3);
 showerrbox := false;
 if GetCheckReal(DP, 3, realout) then
  polyin.gain := realout
 else
  begin
   FrameError(DP, 3, basic1id, showerrbox);
   if not showerrbox then
    SelIText(DP, 3, 0, 255);
  end;
 index := 1;
 for counter := 2 to 11 do
  begin
   realno := 2 * counter;
   imagno := realno + 1;
   GetDData(DP, realno, textreal);
   GetDData(DP, imagno, textimag);
   if ((textreal = '') and (textimag <> '')) then
    begin
     FrameError(DP, imagno, basic1id, showerrbox);
     FrameError(DP, realno, basic1id, showerrbox);
     if not showerrbox then
      SelIText(DP, realno, 0, 255);
    end
   else if GetCheckReal(DP, realno, realout) then
    begin
     polyin.fact[index].realpart := realout;
     polyin.fact[index + 1].realpart := realout;
     order := order + 1;
     if textimag = '' then
      begin
```

```
            polyin.fact[index].imagpart := 0;
            polyin.fact[index].justreal := true;
            index := index + 1;

          end
        else if GetCheckReal(DP, imagno, realout) then
          begin
            order := order + 1;
            polyin.fact[index].imagpart := realout;
            polyin.fact[index].justreal := false;
            polyin.fact[index + 1].imagpart := -realout;
            polyin.fact[index + 1].justreal := false;
            index := index + 2;
          end
        else
          begin
            FrameError(DP, imagno, basic1id, showerrbox);
            if not showerrbox then
              SelIText(DP, imagno, 0, 255);
          end;
      end
    else if not ((textreal = '') and (textimag = '')) then
      begin
        FrameError(DP, realno, basic1id, showerrbox);
        if not showerrbox then
          SelIText(DP, realno, 0, 255);
      end;
  end;
  if order <> polyin.degree then
    begin
      m3 := 'The number of roots does not equal the degree input earlier. Please try again.';
      ParamText(m0, m1, m2, m3);
      SelIText(DP, 4, 0, 255);
      FrameError(DP, 1, basic1id, showerrbox);
    end;
  LoadPolyFactErrorCheck := not showerrbox;
end;
{------------------------------OldCoefDataOut--------------------------------}
procedure OldCoefDataOut;{ ( polyout:polycoef)}
  var
    counter, index : integer;
begin
  SetDData(DP, 3, Real2Str(polyout.gain, true));
  index := polyout.degree + 1;
  for counter := 4 to polyout.degree + 4 do
    begin
      SetDData(DP, counter, Real2Str(polyout.coef[index], true));
      index := index - 1;
    end;
end;
{------------------------------CheckPolyCoef---------------------------------}
procedure CheckPolyCoef; {   polycin:polycoef)}
  var
```

```
    counter, order : integer;
    done : boolean;
begin
  done := false;
  order := polycin.degree + 2;
  repeat
    order := order - 1;
    if polycin.coef[order] <> 0.0 then
      done := true;
  until done or (order = 1);
  polycin.degree := order - 1;
  if (polycin.degree = 0) and (polycin.coef[1] = 0) then
    begin
      polycin.gain := 0;
      polycin.coef[1] := 1;
    end;
end;
{------------------------------LoadPolyCoef--------------------------------}
procedure LoadPolyCoef;
  var
    message : str255;
    index, idno : integer;
    doagain : boolean;
    temppolycoef : polycoef;

begin
  temppolycoef := polyin;
  m0 := textout;
  ParamText(m0, m1, m2, m3);
  index := polyin.degree;
  idno := 7100 + index;
  doagain := true;
  DP := GetNewDialog(idno, nil, pointer(-1));
  if not editnewblock then
    OldCoefDataOut(polyin);
  SelIText(DP, 3, 0, 255);
  while doagain do
    begin
      FrameDItem(DP, 1);
      ModalDialog(nil, itemNum);
      if itemNum = 2 then
        begin
          saveit := false;
          doagain := false
        end
      else
        doagain := not LoadPolyCoefErrorCheck(DP, temppolycoef)
    end;
  if itemNum = 1 then
    begin
      CheckPolyCoef(temppolycoef);
      polyin := temppolycoef;
    end
```

```
    else
      saveit := false;
    DisposDialog(DP);
  end;
{------------------------------OldFactDataOut--------------------------------}
  procedure OldFactDataOut;{  (polyin:polyfact) }
    var
      textout : str255;
      counter, index, rootcount, order : integer;
  begin
    order := polyin.degree;
    counter := 4;
    index := 1;
    rootcount := 0;
    textout := Real2Str(polyin.gain, true);
    SetDData(DP, 3, textout);
    if order > 0 then
      repeat
        if polyin.fact[index].justreal then
          begin
            textout := Real2Str(polyin.fact[index].realpart, true);
            SetDData(DP, counter, textout);
            rootcount := rootcount + 1;
            counter := counter + 2;
            index := index + 1;
          end
        else
          begin
            textout := Real2Str(polyin.fact[index].realpart, true);
            SetDData(DP, counter, textout);
            counter := counter + 1;
            textout := Real2Str(abs(polyin.fact[index].imagpart), true);
            SetDData(DP, counter, textout);
            counter := counter + 1;
            rootcount := rootcount + 2;
            index := index + 2;
          end;
      until rootcount >= order;
  end;
{------------------------------LoadPolyFact----------------------------------}
  procedure LoadPolyFact;
    var
      counter, index : integer;
      doagain : boolean;
      temppolyfact : polyfact;
  begin
    temppolyfact := polyin;
    m0 := textout;
    m1 := Int2Str(polyin.degree);
    m1 := concat('The degree is ', m1);
    ParamText(m0, m1, m2, m3);
    doagain := true;
    DP := GetNewDialog(polyfactid, nil, pointer(-1));
```

```
  If not editnewblock then
    begin
      OldFactDataOut(polyin);
    end;
  while doagain do
    begin
      SelIText(DP, 3, 0, 255);
      FrameDItem(DP, 1);
      ModalDialog(nil, itemNum);
      If itemnum = 2 then
        begin
          saveit := false;
          doagain := false
        end
      else
        doagain := not LoadPolyFactErrorCheck(DP, temppolyfact)
    end;
  If itemnum = 1 then
    polyin := temppolyfact
  else
    saveit := false;
  DisposDialog(DP);
end;
------------------------------ShiftPolyc------------------------------}
procedure ShiftPolyc; {   (polycin:polycoef)}
  var
    shiftby, counter, order : integer;
    done : boolean;
begin
  done := false;
  order := polycin.degree;
  shiftby := 0;
  for counter := order + 1 downto 1 do   { find the amout to shift }
    If polycin.coef[counter] = 0.0 then
      begin
        If not done then
          shiftby := shiftby + 1;
      end
    else
      done := true;
  for counter := order + 1 downto shiftby + 1 do  { shift the coeffs }
    polycin.coef[counter] := polycin.coef[counter - shiftby];
  for counter := shiftby downto 1 do        { set coefs below 'shiftby' = 0 }
    polycin.coef[counter] := 0;
end;
------------------------------GetFactoredData------------------------------}
procedure GetFactoredData;{  var  tempblock:block}
  var
    polyfin : polyfact;
    shiftedpoly : polycoef;
begin
  polyfin.degree := tempblock.num.degree;
  If not editnewblock then
```

```
      begin
        SetCursor(watch);
        shiftedpoly := tempblock.num;
        ShiftPolyc(shiftedpoly);
        DP := GetNewDialog(12466, nil, pointer(-1));
        DrawDialog(DP);
        RootFinder(shiftedpoly, polyfin, true);
        DisposDialog(DP);
        SetCursor(arrow);
      end;
    LoadPolyFact('Numerator Data', polyfin);
    if saveit then
      tempblock.num := FactToCoef(polyfin);
    polyfin.degree := tempblock.den.degree;
    if not editnewblock then
      begin
        SetCursor(watch);
        shiftedpoly := tempblock.den;
        ShiftPolyc(shiftedpoly);
        DP := GetNewDialog(12466, nil, pointer(-1));
        DrawDialog(DP);
        RootFinder(shiftedpoly, polyfin, true);
        DisposDialog(DP);
        SetCursor(arrow);
      end;
    if (saveit or not (editnewblock)) then
      LoadPolyFact('Denominator Data', polyfin);
    if saveit then
      tempblock.den := FactToCoef(polyfin);
  end;

{------------------------------GetCoefData------------------------------}
  procedure GetCoefData; { ( var tempblock:block) }
  begin
    LoadPolyCoef('Numerator Data', tempblock.num);
    if (saveit or not (editnewblock)) then
      LoadPolyCoef('Denominator Data', tempblock.den);
  end;
{------------------------------PutBlockInGroup------------------------------}
  procedure PutBlockInGroup;  {  ( var tempblock:block; var tempgroup:group ) }

    var
      counter : integer;
      done : boolean;
  begin
    done := false;
    if tempblock.factored then
      GetFactoredData(tempblock)
    else
      GetCoefData(tempblock);
    counter := 0;
    if editnewblock and saveit then
      begin
```

```
      tempblock.simplified := false;    { set block parameters}
      tempblock.used := true;
      repeat
        counter := counter + 1;
        if not (tempgroup.bksused[counter]^^.used) then
          begin
            done := true;
            if saveit then
              begin
                tempblock.fromgrpHdl := tempgroup.ownHdl;
                tempgroup.bksused[counter]^^ := tempblock;
                if tempblock.forward then
                  tempgroup.fwdbks := tempgroup.fwdbks + 1
                else
                  tempgroup.backbks := tempgroup.backbks + 1;
                if tempgroup.backbks + tempgroup.fwdbks = 1 then
                  begin
                    ParamText(tempgroup.masterblock^^.title, '', '', '');
                    tempgroup.masterblock^^.simpform := alert(simpformid, nil);
                    ParamText(m0, m1, m2, m3);
                  end;
              end;
          end;
      until done;
    end;
  end;


{------------------------------AddBlockErrorCheck---------------------------}
  function AddBlockErrorCheck;
    const
      addblockerralertid = 16494;
    var
      shownbox : boolean;
      valid : boolean;
      tempreal : extended;
      tempint : integer;
      title : str255;
  begin
    fbackchanged := false;
    shownbox := false;          { set flag - no errors yet    }
    GetDData(DP, 8, Title);     { get title text              }
    If Length(title) > 100 then     { title too long          }
      FrameError(DP, 8, addblockerralertid, shownbox)
    else
      tempblock.title := title;

    GetCheckInt(DP, 9, tempint, valid);
    { change dialog text to integer, check validity}
    If not (valid and (tempint >= 0) and (tempint <= 10)) then
      FrameError(DP, 9, addblockerralertid, shownbox)
    else
      tempblock.num.degree := tempint;
```

```
    GetCheckInt(DP, 10, tempint, valid);
    if not (valid and (tempint >= 0) and (tempint <= 10)) then
      FrameError(DP, 10, addblockerralertid, shownbox)
    else
      tempblock.den.degree := tempint;
    GetDData(DP, 11, path);
    If not ((path = 'f') or (path = 'F') or (path = 'b') or (path = 'B')) then
      FrameError(DP, 11, addblockerralertid, shownbox)
    else if (path = 'f') or (path = 'F') then   { want forward path }
      begin
        If (not editnewblock) and (tempblock.forward = false) then { not new block but changed f-back}
          begin
            with tempgroup do
              begin
                fwdbks := fwdbks + 1;
                backbks := backbks - 1;
              end;
          end;
        tempblock.forward := true;
      end
    else                          { want back path }
      begin
        If (not editnewblock) and (tempblock.forward = true) then  { not new block but changed f-back}
          begin
            with tempgroup do
              begin
                fbackchanged := true;
                fwdbks := fwdbks - 1;
                backbks := backbks + 1;
              end;
          end;
        If editnewblock then
          fbackchanged := true;
        tempblock.forward := false;
      end;
    GetDData(DP, 12, factored);
    If not ((factored = 'f') or (factored = 'F') or (factored = 'c') or (factored = 'C')) then
      FrameError(DP, 12, addblockerralertid, shownbox)
    else if (factored = 'f') or (factored = 'F') then
      tempblock.factored := true
    else
      tempblock.factored := false;

    AddBlockErrorCheck := not shownbox;  {  if error alert box not shown, => no errors }
  end;

{------------------------------DisplayGroup-------------------------}
{  Display the dialog box showing the blocks in the loop   }
{  and return the block that needs to be changed.          }

  procedure DisplayGroup;
{(titleout : str255;var blockoutHdl : bkshdl;var groupout:grpHdl;var showit : bool; exceptsimplified:bool);}
```

```
  var
    fwdno, backno, counter, tempitemtype : integer;
    flagHdl : array[2..11] of bksHdl;
    newblock : block;
    DialogDisposed : boolean;
    temptext  : str255;
    nextgroupout : grpHdl;

begin
  DialogDisposed := false;
  showit := true;
  DP := GetNewDialog(displaygrpid, nil, pointer(-1));
  fwdno := 2;
  backno := 7;
  FrameDItem(DP, 1);
  FrameDItem(DP, 12);    { feedback box }
  FrameDItem(DP, 19);    { simpform box}
  SetDData(DP, 13, titleout);
  for counter := 1 to 5 do
    if groupout^^.bksused[counter]^^.used then
      begin   { load the forward blocks column }
        if groupout^^.bksused[counter]^^.forward then
          begin
            flagHdl[fwdno] := groupout^^.bksused[counter];
            SetDData(DP, fwdno, groupout^^.bksused[counter]^^.title);
            FrameDItem(DP, fwdno);
            fwdno := fwdno + 1;
          end
        else  { load the back blocks column }
          begin
            flagHdl[backno] := groupout^^.bksused[counter];
            SetDData(DP, backno, groupout^^.bksused[counter]^^.title);
            FrameDItem(DP, backno);
            backno := backno + 1;
          end;
      end;
  for counter := fwdno to 6 do
    begin { disable rest of forward path boxes }
      getDitem(DP, counter, itemtype, itemHndl, displayrect);
      tempitemtype := itemtype + 128;
      setDitem(DP, counter, tempitemtype, itemhndl, displayrect);
    end;
  for counter := backno to (11) do
    begin { disable rest of feedback path boxes}
      getDitem(DP, counter, itemtype, itemHndl, displayrect);
      tempitemtype := itemtype + 128;
      setDitem(DP, counter, tempitemtype, itemhndl, displayrect);
    end;

  case groupout^^.masterblock^^.simpform of    { output simp form data }
    1 :
      SetDData(DP, 19, 'Geq');
```

```
    2 :
      SetDData(DP, 19, 'Forward Path');
    3 :
      SetDData(DP, 19, 'Open Loop');
    4 :
      SetDData(DP, 19, 'Closed Loop');
  end;


  if groupout^^.posfback then
    SetDData(DP, 12, ' P')
  else
    SetDData(DP, 12, ' N');
  FrameDItem(DP, 17);
  ModalDialog(nil, itemNum);
  if (itemnum > 1) and (itemnum < 12) then
    begin
      blockoutHdl := flagHdl[itemnum];
      if blockoutHdl^^.simplified then
        begin
          DisposDialog(DP);
          DialogDisposed := true;
          ParamText(blockoutHdl^^.title, '', '', '');
          if not exceptsimplified then
            ignore := 1
          else
            ignore := Alert(blkorgrpid, nil);
          ParamText(m0, m1, m2, m3);
          if ignore = 1 then   { show blocks in that group }
            begin
              nextgroupout := blockoutHdl^^.subgrp;
              temptext := Concat(Omit(titleout, 13, 255), blockoutHdl^^.title);
              DisplayGroup(temptext, blockoutHdl, nextgroupout, showit, exceptsimplified);
            end;
        end;
    end
  else if itemnum = 12 then
    begin
      showit := false;
      ignore := Alert(fbackId, nil);
      if ignore = 1 then
        groupout^^.posFback := false
      else
        groupout^^.posFback := true;
      UpdateData(groupout^^);
    end
  else if itemnum = 19 then   { change simplification form}
    begin
      showit := false;
      ParamText(groupout^^.masterblock^^.title, '', '', '');
      groupout^^.masterblock^^.simpform := Alert(simpformID, nil);
      ParamText(m0, m1, m2, m3);
      UpdateData(groupout^^);
```

```
    end
  else if itemnum = 17 then
    begin
      showit := false;
      DialogDisposed := true;
      DisposDialog(DP);
      EditBlock(groupout^^, newblock, true);
      UpdateData(groupout^^);
    end
  else
    showit := false;
  if not DialogDisposed then
    DisposDialog(DP);
end;
{--------------------------------LoadAddBlockData----------------------------}
procedure LoadAddBlockData;{(blocktochange);}
begin
  SetDData(DP, 8, blocktochange.title);
  SetDData(DP, 9, Int2Str(blocktochange.num.degree));
  SetDData(DP, 10, Int2Str(blocktochange.den.degree));
  if blocktochange.forward then
    SetDData(DP, 11, 'F')
  else
    SetDData(DP, 11, 'B');
  SetDData(DP, 12, 'C');
end;
{--------------------------------CheckBlockGone------------------------------}
procedure CheckBlockGone; {(blockgone : block)}
begin
  with blockgone.fromgrpHdl^^ do
    begin
      if blockgone.forward then          { adjust # of blocks in blockgone's group}
        fwdbks := fwdbks - 1
      else
        backbks := backbks - 1;
      if fwdbks + backbks = 0 then          { was only block in it's group }
        begin
          if not maingrp then
            begin
              masterblock^^.used := false;
              CheckBlockGone(masterblock^^);
            end
          else
            UpdateData(sysgroupH^^);
        end { if = 0 }
      else  { was not only block in group }
        UpdateData(blockgone.fromgrpHdl^^);
    end;{ first  with }
end;
{--------------------------------DeleteBlock---------------------------------}
procedure DeleteBlock;
  var
    grpname : str255;
```

```
    dodelete : boolean;
    blocktodeleteHdl : bksHdl;
    tempgrpH : grpHdl;
  begin
    If (sysgroupH^^.fwdbks + sysgroupH^^.backbks > 0) then
      begin
        tempgrpH := sysgroupH;
        DisplayGroup('Delete from System Group', blocktodeleteHdl, tempgrpH, dodelete, false);
        If (dodelete) then
          begin
            m3 := blocktodeleteHdl^^.title;
            ParamText(m0, m1, m2, m3);
            If (CautionAlert(811, nil) = 2) then
              begin
                blocktodeleteHdl^^.used := false;
                CheckBlockGone(blocktodeleteHdl^^);
              end;
          end;
      end
    else
      Basic1Alert('There are no blocks in the system.', 1);
  end;
{--------------------------------EditBlock-----------------------------}
{ displays dialog box getting info for new block.                       }
  procedure EditBlock;{var grouptochange:group,var blocktochange:block,newblock:boolean)}
    const
      addblockid = 18142;
      OKnum = 1;
      fbackId = 32470;
    var
      doagain : boolean;
      bksingrp : integer;
      msg1, msg2, bksmsg : str255;
  begin
    editnewblock := newblock;
    saveit := true;
    bksingrp := grouptochange.fwdbks + grouptochange.backbks;
    If editnewblock then
      begin
        bksingrp := bksingrp + 1;
      end;
    msg1 := 'Block # ';
    msg2 := int2str(bksingrp);
    bksmsg := ConCat(msg1, msg2);
    If bksingrp > maxbks then
      Basic1Alert('No more blocks can be added to this group. Try simplification.', 3)
    else
      begin
        ClearAllWindows;
        DP := GetNewDialog(addblockid, nil, pointer(-1));
        If not editnewblock then
          begin
            m0 := 'Edit Block Data';
```

```
        LoadAddBlockData(blocktochange);
      end
    else
      begin
        m0 := 'Add New Block Data';
        SetDData(DP, 8, bksmsg);
      end;
    ParamText(m0, m1, m2, m3);
    doagain := true;
    SelIText(DP, 8, 0, 255);
    while doagain do
      begin
        FrameDItem(DP, 1);
        ModalDialog(nil, itemNum);
        if itemNum = 2 then
          doagain := false
        else
          doagain := not AddBlockErrorCheck(DP, grouptochange, blocktochange)
      end;
    DisposDialog(DP);
    if itemNum = OKnum then
      begin
        PutBlockInGroup(blocktochange, grouptochange.ownHdl^^);
        if (not blocktochange.forward) and (grouptochange.backbks = 1) and fbackchanged then
          begin
            ignore := Alert(fbackid, nil);
            if ignore = 1 then
              grouptochange.posfback := false
            else
              grouptochange.posfback := true;
          end;
      end;
  end;
end;

{--------------------------------UpdateData-----------------------------}

procedure UpdateData;     {(groupchanged : group)}
  var
    tempblock, noblock1, noblock2 : block;
begin
  if GeqGroup(groupchanged, groupchanged.masterblock^^.simpform, tempblock, noblock1, noblock2) then
    begin
      groupchanged.masterblock^^.num := tempblock.num;
      groupchanged.masterblock^^.den := tempblock.den;
      if not groupchanged.maingrp then
        UpdateData(groupchanged.masterblock^^.fromgrpHdl^^);
    end
  else
    Basic1Alert('The order of the group blocks are to high to simplify.', 2);
end;
--------------------------------ChangeBlock----------------------------}
procedure ChangeBlock;
```

```
    var
      tempgroupH : grpHdl;
      tempblock : block;
      grpname : str255;
      dochange : boolean;
      blocktoshowHdl : bksHdl;
  begin
    tempgroupH := sysgroupH;
    tempblock := sysblockH^^;
    if (sysgroupH^^.fwdbks + sysgroupH^^.backbks > 0) then   { there are blocks }
      begin
        ParamText(sysblockH^^.title, '', '', '');
        ignore := Alert(blkorgrpid, nil);   { want to see sysblock or sysgroup}
        ParamText(m0, m1, m2, m3);
        if ignore = 2 then              {want sysblock }
          Editblock(tempgroupH^^, tempblock, false)    { data will not be changed }
        else
          begin
            DisplayGroup('Change from System Group', blocktoshowHdl, sysgroupH, dochange, true);
            if dochange then
              begin
                EditBlock(tempgroupH^^, blocktoshowHdl^^, false);
                UpdateData(blocktoshowHdl^^.fromgrpHdl^^);
              end;
          end;
      end
    else
      Basic1Alert('There are no blocks in the system.', 1);
  end;
{--------------------------------DoAddblock-----------------------------}

  procedure DoAddblock;
    var
      tempblock, noblock1, noblock2 : block;
  begin
    EditBlock(sysgroupH^^, tempblock, true);
    if GeqGroup(sysgroupH^^, sysblockH^^.simpform, tempblock, noblock1, noblock2) then
      begin
        sysblockH^^.num := tempblock.num;
        sysblockH^^.den := tempblock.den;
        UpdateData(sysgroupH^^);
      end
    else
      Basic1Alert('The order of the group blocks are to high to simplify.', 2);
  end;

{-------------------------------DoSimplifyGroup---------------------------}
  procedure DoSimplifyGroup;
  begin
    SimpSysGroup;
    UpdateData(sysgroupH^^);
  end;
{-------------------------------ExpandBlock-------------------------------}
```

```
procedure ExpandBlock;
  var
    grpname : str255;
    doexpand : boolean;
    blocktoexpandHdl : bksHdl;
    tempgrpH, newgroupH : grpHdl;
    tempgrpPtr : grpPtr;
    counter : integer;
    stringout : str255;
begin
  if (sysgroupH^^.fwdbks + sysgroupH^^.backbks > 0) then
    begin
      tempgrpH := sysgroupH;
      DisplayGroup('Expand from System Group.', blocktoexpandHdl, tempgrpH, doexpand, false);
      if (doexpand) then
        begin
          newgroupH := grpHdl(NewHandle(Sizeof(group)));
          blocktoexpandHdl^^.simplified := true;
          blocktoexpandHdl^^.simpform := 1;
          blocktoexpandHdl^^.subgrp := newgroupH;
          with newgroupH^^ do
            begin
              ownHdl := newgroupH;
              fwdbks := 1;
              backbks := 0;
              maingrp := false;
              masterblock := blocktoexpandHdl;
              posFback := false;
              for counter := 1 to maxbks do
                begin
                  bksused[counter] := BksHdl(NewHandle(Sizeof(Block)));
                  bksused[counter]^^ := noblock;
                end;{ for }
              bksused[1]^^ := blocktoexpandHdl^^;
              bksused[1]^^.forward := true;
              bksused[1]^^.used := true;
              bksused[1]^^.simplified := false;
              bksused[1]^^.fromgrpHdl := newgroupH;
              blocktoexpandHdl^^.title := Concat(blocktoexpandHdl^^.title, ' Group');
            end;{ with }

          UpdateData(blocktoexpandHdl^^.fromgrpHdl^^);
          stringout := Concat('The block has been replaced with a group titled - ', blocktoexpandHdl^^.title);
          Basic1Alert(stringout, 0);
        end;{ if doexpand }
    end   { if blocks in sysgroup }
  else
    Basic1Alert('There are no blocks in the system.', 1);
end;


nd. { end module }
```

```
unit Bode;
interface
 uses
   XTTypeDefs, Extender1, CADGlobals, NumberCrunch, sane;
 procedure DoBodeMenu;

implementation
 const
   bodedataID = 20645;
   plotpen = 2;
   botmar = 35;
   topmar = 58;
   ltmar = 35;
   rtmar = 35;
   scrnht = 285;
   scrnwd = 493;

 var
   WD : WData;
   bodePic, newPic : PicHandle;
   bodeclipsize, plotrect : rect;
   plotwd, plotht : integer;
   LRC, ULC : point;

{-----------------------GoodBodeDataEntered   procedure-------------------------}

 function GoodBodeDataEntered : boolean;
   var
     tempminfreq, tempmaxfreq, tempminmag, tempmaxmag : integer;
     firsterr, valid, datagood, tempdoit : boolean;
     phaseinc : str255;
 begin
   datagood := true;
   firsterr := true;
   GetCheckInt(DP, 9, tempminfreq, valid);    { check minfreq  }
   If not valid then
     begin
       datagood := false;
       FrameDataError(firsterr, 9);
     end;
   GetCheckInt(DP, 10, tempmaxfreq, valid);     { check maxfreq }
   If ((not valid) or (tempmaxfreq <= tempminfreq)) then
     begin
       datagood := false;
       FrameDataError(firsterr, 10);
     end;
   GetCheckInt(DP, 13, tempminmag, valid);     { check minmag  }
   If not valid then
     begin
       datagood := false;
       FrameDataError(firsterr, 13);
     end;
   GetCheckInt(DP, 14, tempmaxmag, valid);    { check maxfreq }
```

```
  If ((not valid) or (tempmaxmag <= tempminmag)) then
    begin
      datagood := false;
      FrameDataError(firsterr,  14);
    end;
  GetDData(DP, 16, phaseinc);        { Check to include phase plot  }
  If (phaseinc = 'y') or (phaseinc = 'Y') then
    tempdoit := true
  else If (phaseinc = 'n') or (phaseinc = 'N') then
    tempdoit := false
  else
    begin
      datagood := false;
      FrameDataError(firsterr,  16);
    end;
  GoodBodeDataEntered := datagood;

  If datagood then
    with bodedata do
      begin
        minfreq  := tempminfreq;
        maxfreq  := tempmaxfreq;
        minmag := tempminmag;
        maxmag := tempmaxmag;
        doit := tempdoit;
      end;
end;

{-----------------------InitBodeData    procedure-------------------------}

procedure InitBodeData;
begin
  DP := GetNewDialog(bodedataID, nil, pointer(-1));
  SelIText(DP, 9, 0, 255);      { select the min freq input block  }
  with bodedata do
    begin              { initialize the dialog data }
      SetDData(DP, 9, Int2Str(minfreq));
      SetDData(DP, 10, Int2Str(maxfreq));
      SetDData(DP, 13, Int2Str(minmag));
      SetDData(DP, 14, Int2Str(maxmag));
      If doit then
        SetDData(DP, 16, 'Y')
      else
        SetDData(DP, 16, 'N');
    end;
end;
{-----------------------GetBodeData    procedure-------------------------}

procedure GetBodeData (var continue : boolean);
  var
    doagain : boolean;
begin
  continue := true;
```

```
    doagain := true;              { init flag to show dialog box }
    InitBodeData;                 { set up bode dialog box    }
    while doagain do
      begin
        FrameDItem(DP, 1);
        ModalDialog(nil, itemNum);
        if itemNum = 2 then
          begin
            continue := false;
            doagain := false;
          end
        else
          begin
            ClearAllWindows;
            doagain := not GoodBodeDataEntered;
          end;
      end;
    DisposDialog(DP);
  end;

{------------------------Wd2Freq    function--------------------------}
{ input the horizontal position and the appropriate frequency is returned  }
  function Wd2Freq (hpos : integer) : extended;

  begin
    with bodedata do
      Wd2Freq := Ten2((maxfreq - minfreq) / plotwd * (hpos - ltmar) + minfreq);
  end;

{------------------------Freq2Wd    function--------------------------}
  function Freq2Wd (freq : extended;
                    var wd : integer) : boolean;
    var
      inbounds : boolean;
      freqpos : extended;
  begin
    inbounds := true;
    freqpos := log(freq);
    with bodedata do
      begin
        if (maxfreq > freqpos) and (freqpos > minfreq) then
          begin
            wd := ltmar + Num2Integer((freqpos - minfreq) * plotwd / (maxfreq - minfreq));
          end
        else
          inbounds := false;
      end;
    Freq2Wd := inbounds;
  end;

{------------------------Mag2Ht    function--------------------------}
  function Mag2Ht (mag : extended;
                   var ht : integer) : boolean;
```

```
  var
    inbounds : boolean;
begin
  inbounds := true;
  with bodedata do
    begin
      if (maxmag >= mag) and (mag >= minmag) then
        ht := topmar + Num2Integer((maxmag - mag) * plotht / (maxmag - minmag))
      else
        inbounds := false;
    end;
  Mag2Ht := inbounds;
end;
{-----------------------DrawHorzLine    procedure-------------------------}
procedure DrawHorzLine;
  const
    initscale = 7;
  var
    scale, magline, mag : integer;
    strout : str255;
begin
  scale := FindSep(bodedata.maxmag, bodedata.minmag, initscale);   { change to no. of units per division}
  mag := bodedata.minmag;
  repeat
    if Mag2Ht(mag, magline) then
      begin
        strout := Int2Str(mag);
        if (bodedata.minmag < mag) and (bodedata.maxmag > mag) then
          DrawLine(ltmar, magline, (scrnwd - rtmar - 1), magline);
        MoveTo(ltmar - 5 - StringWidth(strout), magline + 5);
        DrawString(strout);
      end;
    mag := mag + scale;
  until mag > (bodedata.maxmag);
  if Mag2Ht(0, magline) then
    begin
      pensize(2, 2);
      DrawLine(ltmar, magline, (scrnwd - rtmar - 2), magline);
      pensize(1, 1);
    end;
end;
-----------------------DrawVertLine    procedure-------------------------}
procedure DrawVertLine (freq : extended);
  var
    horspos : integer;
begin
  if Freq2Wd(freq, horspos) then
    Drawline(horspos, topmar, horspos, (topmar + plotht - 1));
end;

-----------------------Freq2Str    function-------------------------}
change frequency from its exponent to a string with min length       }
```

```
function Freq2Str (expo : integer) : str255;
  var
    strout : str255;
    counter : integer;
begin
  strout := '1';
  if expo > 0 then     { if freq is greater than 1 }
    for counter := 1 to expo do
      strout := concat(strout, '0');
  if expo < 0 then     { if freq is less than 1 }
    begin
      for counter := 1 to -expo do
        if counter > 1 then        { skips first zero if exp is -1 }
          strout := concat('0', strout);
      strout := concat('.', strout)
    end;
  if (expo < -3) or (expo > 3) then
    strout := concat('1 e', Int2Str(expo));
  Freq2Str := strout;
end;

{-----------------------PlotMag    procedure-------------------------}

procedure PlotMag;
  const
    freqposstep = 2;
    twopi = 6.2831853;
  var
    freq : extended;
    oldmagpoint, newmagpoint, oldphasepoint, newphasepoint : point;
    tempmag, tempphase : extended;
    maght, freqpos : integer;
    wasinplot, firstpoint : boolean;
begin
  ClipRect(plotrect);
  firstpoint := true;
  wasinplot := false;
  freqpos := ltmar - freqposstep;
  DP := GetNewDialog(calcpointid, nil, pointer(-1));
  SetDData(DP, 1, 'Calculating data points. Please be patient!');
  SetDData(DP, 2, Int2Str(plotwd + ltmar));
  repeat
    pensize(2, 2);
    freqpos := freqpos + freqposstep;
    freq := Wd2Freq(freqpos);
    tempmag := 20.0 * log(EvalGeq(sysblockH^^, freq, tempphase));
    if Mag2Ht(tempmag, maght) then
      begin
        newmagpoint.h := freqpos;
        newmagpoint.v := maght;
        if wasinplot then          { last value was displayed in plot }
          begin
            DrawLine(oldmagpoint.h, oldmagpoint.v, newmagpoint.h, newmagpoint.v);
```

```
          DrawLine(oldmagpoint.h, oldmagpoint.v, newmagpoint.h, newmagpoint.v);
          oldmagpoint := newmagpoint;
        end
      else
        begin
          oldmagpoint := newmagpoint;
          wasinplot := true;
        end;
    end;
  tempphase := -tempphase;    { change to a positive angle for plotting }
  while tempphase < 0 do
    tempphase := twopi + tempphase;
  while tempphase > twopi do
    tempphase := tempphase - twopi;
  newphasepoint.h := freqpos;
  newphasepoint.v := topmar + num2integer(tempphase * plotht / twopi);
  pensize(3, 3);
  if bodedata.doit then
    begin
      if firstpoint then
        firstpoint := false
      else
        DrawLine(oldphasepoint.h, oldphasepoint.v, newphasepoint.h, newphasepoint.v);
    end;
  oldphasepoint := newphasepoint;
  SetDData(DP, 2, Int2Str(plotwd + ltmar - freqpos));
until freqpos >= plotwd + ltmar;
DisposDialog(DP);
pensize(1, 1);
ClipRect(bodeclipsize);
end;
{-----------------------DrawBasicPlot    procedure-------------------------}
procedure DrawBasicPlot;
  var
    index, counter, numpos : integer;
    numout : str255;
    freq : extended;
begin
  plotrect.topleft := ULC;        {  outline for plot  }
  plotrect.botright := LRC;
  pensize(plotpen, plotpen);
  penpat(black);
  framerect(plotrect);
  pensize(1, 1);
  DrawHorzLine;
  begin
    MoveTo(ltmar, (topmar + plotht + 12));        {  draw first freq label }
    DrawString(Freq2Str(bodedata.minfreq));
  end;
  freq := Ten2(bodedata.minfreq);
  for index := bodedata.minfreq to bodedata.maxfreq do
    begin
      if Freq2Wd(freq, numpos) then
```

```
        begin
          MoveTo(numpos, (topmar + plotht + 12));        {  draw freq below axis  }
          DrawString(Freq2Str(index));
        end;
      for counter := 1 to 9 do
        if index < bodedata.maxfreq then
          begin
            freq := counter * Ten2(index);
            DrawVertLine(freq);
          end;
      MoveTo(ltmar + Num2Integer((plotwd - StringWidth('Frequency (Rads/sec)')) / 2), scrnht - 11);
      DrawString('Frequency (Rads/sec)');
    end;
  PenNormal;
end;
{-----------------------LabelPhase   procedure-------------------------}
procedure LabelPhase;
  const
    hoffset = 25;
    voffset = 0;
  var
    hpos, vpos, sep, counter : integer;
begin
  sep := plotht div 12;
  for counter := 0 to 12 do
    begin
      penpat(gray);
      vpos := topmar + counter * sep;
      if (0 < counter) and (counter < 12) then
        DrawLine(ltmar, vpos, ltmar + plotwd - 1, vpos);
      case counter of
        0 :
          begin
            moveto(ltmar + plotwd + 3, topmar + voffset);
            penpat(black);
            DrawString('0');
          end;
        3 :
          begin
            move(3, voffset);
            penpat(black);
            DrawString('-90');
          end;
        6 :
          begin
            move(3, voffset);
            DrawString('-180');
          end;
        9 :
          begin
            move(3, voffset);
            penpat(black);
            DrawString('-270');
```

```
      12 :
        begin
          moveto(ltmar + plotwd + 3, topmar + plotht + voffset);
          penpat(black);
          DrawString('-360');
        end;
      otherwise
        ;
    end;
  end;
end;
{-----------------------DoBodeMenu   procedure-------------------------}

procedure DoBodeMenu;
  var
    dotheplot : boolean;
begin
  plotht := scrnht - topmar - botmar;
  plotwd := scrnwd - ltmar - rtmar;
  LRC.v := topmar + plotht;
  LRC.h := ltmar + plotwd;
  ULC.v := topmar;
  ULC.h := ltmar;
  SetRect(bodeclipsize, 0, 0, 512, 323);
  itemnum := alert(bodeselID, nil);
  TextFace([bold]);
  case itemnum of
    1 :       { seelect redraw }
      begin
        if (sysgroupH^^.fwdbks + sysgroupH^^.backbks > 0) then
          begin
            if bodedata.layer > 0 then
              begin
                ShowWindow(bodePtr);
                SelectWindow(bodePtr);
              end
            else
              Basic1Alert('A Bode plot has not yet been drawn.', 1);
          end
        else
          Basic1Alert('There are no blocks in the system.', 1);
      end;
    2 :       { select new plot }
      begin
        if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
          begin
            ClearAllWindows;
            GetBodeData(dotheplot);
            if dotheplot then
              begin
                ShowWindow(bodePtr);
                SelectWindow(bodePtr);
                ClipRect(bodeclipsize);
```

```
            bodePic := OpenPicture(bodeclipsize);
            SetCursor(watch);
            LabelPhase;
            DrawBasicPlot;
            PlotMag;
            PenNormal;
            ClipRect(bodeclipsize);
            ClosePicture;
            SetWPic(bodePtr, bodePic);
            SetCursor(arrow);
            bodedata.layer := 1;
          end;
      end
    else
      Basic1Alert('There are no blocks in the system.', 1);
  end;
3 :     { select overlap }
  begin
    if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
      begin
        if bodedata.layer > 0 then
          begin
            bodedata.layer := bodedata.layer + 1;
            HideWindow(bodePtr);
            ShowWindow(bodePtr);
            SelectWindow(bodePtr);
            ClipRect(bodeclipsize);
            GetWData(bodePtr, WD);
            bodePic := WD.windowPic;
            newPic := OpenPicture(bodeclipsize);
            DrawPicture(bodePic, bodeclipsize);
            pensize(plotpen, plotpen);
            penpat(black);
            SetCursor(watch);
            case bodedata.layer of
              2 :
                penpat(dkgray);
              3 :
                penpat(gray);
              otherwise
                penpat(ltgray);
            end;
            PlotMag;
            SetCursor(arrow);
            ClosePicture;
            PenNormal;
            SetWPic(bodePtr, newPic);
            ShowWindow(bodePtr);
            SelectWindow(bodePtr);
          end
        else
          Basic1Alert('This would be the first plot.', 1);
      end
```

```
      else
        Basic1Alert('There are no blocks in the system.', 1);
      end;
    otherwise
      ;
  end;{ case}
 end;
end.
```

```
unit Nyquist;
interface
  uses
    XTTypeDefs, Extender1, CADGlobals, NumberCrunch, SANE, Extend2Stuff;
  procedure DoNyquistMenu;

implementation
  const
    pi = 3.141592658;
    plotpen = 2;
    scrnwd = 493;
    scrnht = 285;
    topmar = 15;
    ltmar = 38;
    botmar = 15;
  type
    pointdata = record
        donedata, wasabove, isabove : boolean;
        phasept, freqpt, magpt : extended;
      end;
  var
    plotorigh, plotorigv : integer;
    nyqPic, newPic : PicHandle;
    clipsize : rect;
    multfactor : extended;
    phasemarg, gainmarg, magthreehalf, maghalf, magtwo, magthree : pointdata;

{-------------------------PointInPlot   function---------------------}
  function PointInPlot (point1, point2 : point) : boolean;
    var
      inplot : boolean;
      point1mag, point2mag : extended;
  begin
    inplot := true;
    point1mag := Sqrt(Num2Extended(point1.v) * Num2Extended(point1.v) + Num2Extended(point1.h) *
        Num2Extended(point1.h));
    point2mag := Sqrt(Num2Extended(point2.v) * Num2Extended(point2.v) + Num2Extended(point2.h) *
        Num2Extended(point2.h));
    if (point1mag > radius) and (point2mag > radius) then
      inplot := false;
    PointInPlot := inplot;
  end;

{-------------------------Pol2Rec   procedure---------------------}
  procedure Pol2Rec (mag, ang : extended;
              var x : extended;
              var y : extended);
  begin
    x := mag * cos(ang);
    y := mag * sin(ang);
  end;

{---------------------DoRadialGrid   procedure------------------}
```

```
procedure DoRadialGrid;
  const
    initscale = 4;
  var
    scale, counter, divint : integer;
    divs : extended;
    xpos, ypos : integer;
    strout : str255;
begin
  with nyquistdata do
    begin
      scale := maxmag div FindSep(maxmag, 0, initscale);
      divs := multfactor * maxmag / scale;  { adjusted radial value in pixels between separations  }
      divint := Num2Integer(maxmag / scale);  { integer value of interval  }
    end;
  pensize(1, 1);
  for counter := 1 to scale do
    begin
      penpat(black);
      xpos := Num2Integer((counter) * divs);
      ypos := -3;
      MoveTo(plotorigh + xpos, plotorigv + ypos);
      strout := Int2Str((counter * divint));
      DrawString(strout);
      penpat(dkgray);
      FrameCircle(plotorigh, plotorigv, counter * divs);
    end;
  if nyquistdata.maxmag = 1 then
    begin
      penpat(dkgray);
      for counter := 1 to 9 do
        FrameCircle(plotorigh, plotorigv, counter * radius / 10);
    end;
end;

----------------------DrawBasicPlot   procedure-----------------}
procedure DrawBasicPlot;
begin
  penpat(dkgray);
  pensize(1, 1);      { draw X and Y axis  }
  DrawLine(-radius + plotorigh, plotorigv, radius + plotorigh, plotorigv);
  DrawLine(plotorigh, plotorigv - radius, plotorigh, plotorigv + radius - 1);
  DoRadialGrid;
  penpat(black);
  FrameCircle(plotorigh, plotorigv, multfactor);
  MoveTo(plotorigh + Num2Integer(multfactor) + 1, plotorigv - 3);
  WriteDraw('1');
  PenSize(plotpen, plotpen);
  FrameCircle(plotorigh, plotorigv, radius);
draw outer plot circle }

  MoveTo(radius + 1 + plotorigh, 14 + plotorigv);    { label phase positions }
  WriteDraw('0');
```

```
    MoveTo(plotorigh - 14, plotorigv + radius + 11);
    WriteDraw('-90');
    MoveTo(plotorigh - radius - 36, plotorigv + 4);
    WriteDraw('-180');
    MoveTo(plotorigh - 19, plotorigv - radius - 2);
    WriteDraw('-270');
  end;

{----------------------GetPointData   procedure-----------------}
  procedure GetPointData (var pointofinterest : pointdata;
              magvalue, magofinterest, phaseofinterest, freqofinterest : extended);
  begin
    with pointofinterest do          { get mag data }
      begin
        if not donedata then
          begin
            if magofinterest > magvalue then
              isabove := true
            else
              isabove := false;
            if wasabove and not isabove then
              begin
                phasept := phaseofinterest * 180 / pi;
                freqpt  := freqofinterest;
                donedata := true;
              end;  { save data   }
            wasabove := isabove;
          end;
      end;{ with }
  end;

{----------------------WriteFreq    procedure-----------------}
  procedure WriteFreq (freqout : extended);
    var
      multiplier : extended;
      midmove, extrememove : integer;
  begin
    midmove := -StringWidth('0');    { set interval to move for each decimal place to left }
    extrememove := 2 * midmove;
    if (freqout < 1e-3) or (freqout > 1e3) then
      begin
        Move(extrememove, 0);
        WriteDraw(freqout);
      end
    else
      begin
        multiplier := 10;
        while freqout >= multiplier do
          begin
            Move(midmove, 0);
            multiplier := multiplier * 10;
          end;
        WriteDraw(freqout : 3 : 3);
```

```
    end;
 end;

{---------------------DoDataBox   procedure-----------------}
 procedure DoDataBox;
  const
    lettersize = 9;
    upmar = 0;
    lowmar = 7;
    gainmarmar = 10;
    ltind = 7;
    rtind = 7;
    lineht = 16;
    linewd = 170;
    magind = 15;
    phaseind = 80;
    freqind = 145;
    extraind = 0;
  var
    ulc, lrc : point;
    boxwd, boxht, totlines, templines, marginlines, datalines, maxboxht, boxcenter : integer;
    maxbox, databox, temprect : rect;
    doupper, dolower : boolean;
 begin
  TextFace([bold]);
  penpat(black);
  doupper := false;
  dolower := false;
  lrc.h := scrnwd - 3;
  lrc.v := scrnht - 3;
  boxwd := ltind + rtind + linewd;
  ulc.h := lrc.h - boxwd;
  maxboxht := 8 * lineht + upmar + lowmar;
  boxcenter := ulc.h + Num2Integer(boxwd / 2);
  SetRect(maxbox, ulc.h, lrc.v - maxboxht, lrc.h, lrc.v); { define rect to erase old data}
  EraseRect(maxbox);
  databox := maxbox;
  marginlines := 0;
  datalines := 1;
  If phasemarg.donedata then  { margin points ,upper data }
    begin
      marginlines := marginlines + 1;
      datalines := datalines + 1;
    end;
  If gainmarg.donedata then
    marginlines := marginlines + 1;
  If magthreehalf.donedata then { data points  lowerdata   }
    datalines := datalines + 1;
  If maghalf.donedata then
    datalines := datalines + 1;
  If magtwo.donedata then
    datalines := datalines + 1;
  If magthree.donedata then
```

```
    datalines := datalines + 1;
  if marginlines > 0 then      { is upper data needed }
    doupper := true;
  if datalines > 1 then        { is lower data needed }
    dolower := true;
  if doupper or dolower then
    begin      { do the data box }
      totlines := 0;
      if doupper then
        totlines := marginlines;
      if dolower then
        totlines := totlines + datalines;
      boxht := upmar + lowmar + totlines * lineht;
      ulc.v := lrc.v - boxht;      { set last box coordinate}
      SetRect(databox, ulc.h, ulc.v, lrc.h, lrc.v);
      Pensize(1, 1);
      FrameRect(databox);
      PenSize(2, 2);
      temprect := databox;
      InsetRect(temprect, 2, 2);
      FrameRect(temprect);
      Pensize(1, 1);
      TextSize(lettersize);
      templines := 0;
      if gainmarg.donedata then      { do gain margin data}
        begin
          templines := templines + 1;              { add a line }
          MoveTo(ulc.h + gainmarmar, ulc.v + upmar + lineht * templines);
          DrawString('Gain Margin (dB) = ');
          WriteDraw(gainmarg.magpt : 3 : 2);
        end;
      if phasemarg.donedata then     { do phase margin data  }
        begin
          templines := templines + 1;              { add a line }
          MoveTo(ulc.h + gainmarmar, ulc.v + upmar + lineht * templines);
          DrawString('Phase Margin (deg) = ');
          WriteDraw(180 - phasemarg.phasept : 3 : 2);
        end;
      if doupper then      { draw a line separating upper and lower data box  }
        DrawLine(ulc.h + 2, ulc.v + upmar + templines * lineht + 2, lrc.h - 3, ulc.v + upmar + templines *
      lineht + 2);
      if dolower then      { do lower data box  }
        begin
          templines := templines + 1;              { add a line }
          MoveTo(ulc.h + magind, ulc.v + upmar + templines * lineht);  { draw column titles }
          DrawString('Mag');
          MoveTo(ulc.h + phaseind, ulc.v + upmar + templines * lineht);
          DrawString('Phase');
          MoveTo(ulc.h + freqind, ulc.v + upmar + templines * lineht);
          DrawString('Freq');
          DrawLine(ulc.h + 2, ulc.v + upmar + templines * lineht + 2, lrc.h - 3, ulc.v + upmar + templines *
      lineht + 2);
          if maghalf.donedata then      { do mag half point }
```

```
        begin
          templines := templines + 1;              { add a line }
          MoveTo(ulc.h + magind, ulc.v + upmar + templines * lineht);
          DrawString('0.5');
          MoveTo(ulc.h + phaseind, ulc.v + upmar + templines * lineht);
          WriteDraw(maghalf.phasept : 3 : 1);
          MoveTo(ulc.h + freqind + extraind, ulc.v + upmar + templines * lineht);
          WriteFreq(maghalf.freqpt);
        end;  { mag half}
      if phasemarg.donedata then      { do mag one point }
        begin
          templines := templines + 1;              { add a line }
          MoveTo(ulc.h + magind, ulc.v + upmar + templines * lineht);
          DrawString('1.0');
          MoveTo(ulc.h + phaseind, ulc.v + upmar + templines * lineht);
          WriteDraw(-phasemarg.phasept : 3 : 1);
          MoveTo(ulc.h + freqind + extraind, ulc.v + upmar + templines * lineht);
          WriteFreq(phasemarg.freqpt);
        end;  { mag one }
      if magthreehalf.donedata then      { do mag threehalf point }
        begin
          templines := templines + 1;
          MoveTo(ulc.h + magind, ulc.v + upmar + templines * lineht);
          DrawString('1.5');
          MoveTo(ulc.h + phaseind, ulc.v + upmar + templines * lineht);
          WriteDraw(magthreehalf.phasept : 3 : 1);
          MoveTo(ulc.h + freqind + extraind, ulc.v + upmar + templines * lineht);
          WriteFreq(magthreehalf.freqpt);
        end;  { mag threehalf}
      if magtwo.donedata then      { do mag two point }
        begin
          templines := templines + 1;
          MoveTo(ulc.h + magind, ulc.v + upmar + templines * lineht);
          DrawString('2.0');
          MoveTo(ulc.h + phaseind, ulc.v + upmar + templines * lineht);
          WriteDraw(magtwo.phasept : 3 : 1);
          MoveTo(ulc.h + freqind + extraind, ulc.v + upmar + templines * lineht);
          WriteFreq(magtwo.freqpt);
        end;  { mag two}
      if magthree.donedata then      { do mag three point }
        begin
          templines := templines + 1;
          MoveTo(ulc.h + magind, ulc.v + upmar + templines * lineht);
          DrawString('3.0');
          MoveTo(ulc.h + phaseind, ulc.v + upmar + templines * lineht);
          WriteDraw(magthree.phasept : 3 : 1);
          MoveTo(ulc.h + freqind + extraind, ulc.v + upmar + templines * lineht);
          WriteFreq(magthree.freqpt);
        end;  { mag half}
    end;          { end lower data box }
  end;      { do the data box }
TextSize(12);
Pensize(1, 1);
```

```
  end;

{----------------------DataToGraph   procedure-----------------}
  procedure DataToGraph;
    var
      counter : integer;
      premag, adjustedmag, phase : extended;
      freq, freqinterval : extended;
      oldpoint, newpoint : point;
      decades : extended;
{ freqinterval is the segment of the log scale over the freq range. }
  begin
    SetClip(plotclipH);
    phasemarg.donedata := false;     { set points of interest data }
    phasemarg.wasabove := false;
    gainmarg.donedata := false;
    gainmarg.wasabove := false;
    magthreehalf.donedata := false;
    magthreehalf.wasabove := false;
    maghalf.donedata := false;
    maghalf.wasabove := false;
    magtwo.donedata := false;
    magtwo.wasabove := false;
    magthree.donedata := false;
    magthree.wasabove := false;
    decades := Log(nyquistdata.maxfreq) - Log(nyquistdata.minfreq);
    If nyquistdata.linear then
      freqinterval := (nyquistdata.maxfreq - nyquistdata.minfreq) / nyquistdata.pointstoplot
    else
      freqinterval := decades / nyquistdata.pointstoplot;
    DP := GetNewDialog(calcpointid, nil, pointer(-1));
    SetDData(DP, 1, 'Calculating data points. Please be patient!');
    SetDData(DP, 2, Int2Str(nyquistdata.pointstoplot));
    for counter := 1 to nyquistdata.pointstoplot do
      begin
        If nyquistdata.linear then
          freq := nyquistdata.minfreq + counter * freqinterval
        else
          freq := Ten2(Log(nyquistdata.minfreq) + counter * freqinterval);
        premag := EvalGeq(sysblockH^^, freq, phase);
        with phasemarg do          { get phase margin data }
          begin
            If not donedata then
              begin
                If premag > 1 then
                  isabove := true
                else
                  isabove := false;
                If wasabove and not isabove then
                  begin
                    phasept := -phase * 180 / pi;
                    freqpt := freq;
                    donedata := true;
```

```
            end;  { save data   }
          wasabove := isabove;
        end;
      end;{ with phasemarg}
    GetPointData(maghalf, 0.5, premag, phase, freq); { points of interest}
    GetPointData(magthreehalf, 1.5, premag, phase, freq);
    GetPointData(magtwo, 2.0, premag, phase, freq);
    GetPointData(magthree, 3.0, premag, phase, freq);

    with gainmarg do          { get gain margin data }
      begin
        if not donedata then
          begin
            if phase > -pi then
              isabove := true
            else
              isabove := false;
            if wasabove and not isabove then
              begin
                magpt := -20 * log(premag);
                freqpt := -freq;
                donedata := true;
              end;  { save data   }
            wasabove := isabove;
          end;
      end;{ with gainmarg  }

    adjustedmag := multfactor * premag;
    Pole2Rect(adjustedmag, phase, newpoint.h, newpoint.v);
    if (counter > 1) and PointInPlot(oldpoint, newpoint) then
      Drawline(oldpoint.h + plotorigh, oldpoint.v + plotorigv, newpoint.h + plotorigh, newpoint.v +
      plotorigv);
    oldpoint := newpoint;
    SetDData(DP, 2, Int2Str(nyquistdata.pointstoplot - counter));
  end;
  DisposDialog(DP);
  ClipRect(clipsize);
  DoDataBox;
  ClipRect(clipsize);
end;

----------DrawNewPlot   procedure----------}
procedure DrawNewPlot;
begin
  if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
    begin
      SetCursor(watch);
      ShowWindow(NyqPtr);
      SelectWindow(NyqPtr);
      TextFace([bold]);
      ClipRect(clipsize);
      NyqPic := OpenPicture(clipsize);
      PenSize(plotpen, plotpen);
```

```
      DrawBasicPlot;
      DataToGraph;
      PenNormal;
      ClosePicture;
      SetWPic(nyqPtr, nyqPic);
      SetCursor(arrow);
    end
  else
    Basic1Alert('There are no blocks in the system.', 1);
end;

{-----------------GoodNyquistDataEntered  function--------------}
function GoodNyquistDataEntered : boolean;
  var
    tempminfreq, tempmaxfreq : extended;
    tempmaxmag, temppointstoplot : integer;
    firsterr, valid, datagood : boolean;
begin
  datagood := true;
  firsterr := true;
  GetCheckInt(DP, 7, tempmaxmag, valid);     { check max mag }
  if (not valid) or (tempmaxmag <= 0) then
    begin
      datagood := false;
      FrameDataError(firsterr, 7);
    end;
  valid := GetCheckReal(DP, 8, tempminfreq);    { check min freq }
  if (not valid) then
    begin
      datagood := false;
      FrameDataError(firsterr, 8);
    end;
  valid := GetCheckReal(DP, 9, tempmaxfreq);     { check max freq }
  if (not valid) or (tempminfreq >= tempmaxfreq) then
    begin
      datagood := false;
      FrameDataError(firsterr, 9);
    end;
  GetCheckInt(DP, 12, temppointstoplot, valid);    { check max mag }
  if (not valid) or (temppointstoplot <= 0) then
    begin
      datagood := false;
      FrameDataError(firsterr, 12);
    end;
  GoodNyquistDataEntered := datagood;
  if datagood then
    with nyquistdata do
      begin
        minfreq := tempminfreq;
        maxfreq := tempmaxfreq;
        maxmag := tempmaxmag;
        pointstoplot := temppointstoplot;
        multfactor := radius / maxmag;
```

```
    end;
end;

{----------RedrawPlot  procedure----------}
procedure RedrawPlot;
begin
  if not (sysgroupH^^.fwdbks + sysgroupH^^.backbks > 0) then
    Basic1Alert('There are no blocks in the system.', 1)
  else
    begin
      if not (nyquistdata.layer > 0) then
        Basic1Alert('A Nyquist plot has not yet beed drawn.', 1)
      else
        begin
          ShowWindow(nyqPtr);
          SelectWindow(nyqPtr);
        end;
    end;
end;

{----------GetNyquistData  procedure----------}
procedure GetNyquistData;
  var
    doagain, templinear : boolean;
begin
  templinear := nyquistdata.linear;
  doagain := true;
  SellText(DP, 7, 0, 255);
  while doagain do
    begin
      FrameDItem(DP, 1);
      ModalDialog(nil, itemNum);
      case itemnum of
        1 :       { selected OK }
          begin
            doagain := not GoodNyquistDataEntered;
            if not doagain then
              begin
                nyquistdata.linear := templinear;
                DisposDialog(DP);
                DrawNewPlot;
                nyquistdata.layer := 1;
              end;
          end;
        3 :     { selected Cancel is item 3 }
          begin
            DisposDialog(DP);
            doagain := false;
          end;
        13 :    { log button }
          begin
            if templinear then
              begin
```

```
            CheckDItem(DP, 13);
            CheckDItem(DP, 14);
            templinear := false;
          end;
        end;
      14 :    { linear button }
        begin
          if not templinear then
            begin
              CheckDItem(DP, 13);
              CheckDItem(DP, 14);
              templinear := true;
            end;
        end;
    end; {case}
  end;
end;

{----------------------InitNyquistDialog   procedure-----------------}
procedure InitNyquistDialog;
begin
  DP := GetNewDialog(nyquistid, nil, pointer(-1));
  SelIText(DP, 7, 0, 255);
  with nyquistdata do
    begin
      SetDData(DP, 7, Int2Str(maxmag));
      SetDData(DP, 8, Real2Str(minfreq, true));
      SetDData(DP, 9, Real2Str(maxfreq, true));
      SetDData(DP, 12, Int2Str(pointstoplot));
      if linear then
        CheckDItem(DP, 14)        { set linear button }
      else
        checkDItem(DP, 13);
    end;
end;

{----------------------InitPlotStuff   procedure-----------------}
procedure InitPlotStuff;
begin
  radius := Num2Integer((scrnht - topmar - botmar) / 2);
  plotorigh := (radius + ltmar);
  plotorigv := (radius + topmar);
  plotclipH := NewRgn;
  OpenRgn;
  FrameCircle(plotorigh, plotorigv, radius);
  CloseRgn(plotclipH);
    { set clipsize to screen size after the origin has been shifted }
  SetRect(clipsize, 0, 0, 512, 323);
  firstnyquistrun := false;
end;

{----------------------DoNyquistMenu   procedure-----------------}
procedure DoNyquistMenu;
```

```
begin
  if firstnyquistrun then
    InitPlotStuff;
  itemnum := alert(nyqalertid, nil);
  case itemnum of
    1 :       { selected Redraw }
      begin
        RedrawPlot;
      end;
    2 :       { selected New Plot }
      begin
        if not (sysgroupH^^.fwdbks + sysgroupH^^.backbks > 0) then
          Basic1Alert('There are no blocks in the system.', 1)
        else
          begin
            ClearAllWindows;
            InitNyquistDialog;
            GetNyquistData;
          end;
      end;
    3 :       { select overlap plot }
      begin
        if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
          begin
            if nyquistdata.layer > 0 then
              begin
                nyquistdata.layer := nyquistdata.layer + 1;
                HideWindow(nyqPtr);
                ShowWindow(nyqPtr);
                SelectWindow(nyqPtr);
                ClipRect(clipsize);
                nyqPic := GetWPic(nyqPtr);
                newPic := OpenPicture(clipsize);
                DrawPicture(nyqPic, clipsize);
                pensize(plotpen, plotpen);
                penpat(black);
                SetCursor(watch);
                case nyquistdata.layer of
                  2 :
                    penpat(dkgray);
                  3 :
                    penpat(gray);
                  otherwise
                    penpat(ltgray);
                end;
                DataToGraph;
                SetCursor(arrow);
                ClosePicture;
                PenNormal;
                SetWPic(nyqPtr, newPic);
                penpat(black);
                ShowWindow(nyqPtr);
                SelectWindow(nyqPtr);
```

```
            end
          else
            Basic1Alert('This would be the first plot.', 1);
        end
      else
        Basic1Alert('There are no blocks in the system.', 1);
    end;
  otherwise      { selected Cancel }
    ;
  end; {case}
 end;
end.
```

```
unit RootLocus;
interface
 uses
   xttypedefs, extender1, CadGlobals, NumberCrunch, simpgroup, SANE, Extend2Stuff;

 procedure DoRLocusMenu;
implementation
 const
   botmar = 20;
   topmar = 58;
   ltmar = 15;
   rtmar = 55;
   scrnht = 285;
   scrnwd = 493;
 var
   rlocusPic, newPic : PicHandle;
   clipsize, plotrect : rect;
   plotwd, plotht, pointno : integer;
   xstep, ystep, gainval : extended;

{-----------------------   X2Wd   procedure---------------------}
 function X2Wd (xin : extended) : integer;
   var
     tempxout : extended;
     xout : integer;
 begin
   with rlocusdata do
     begin
       if (xmin <= xin) and (xin <= xmax) then
         begin
           tempxout := ltmar + plotwd * (xin - xmin) / (xmax - xmin);
           xout := Num2Integer(tempxout)
         end   { if}
       else
         xout := 512;      { put it outside of plot }
     end;      { with }
   X2Wd := xout;
 end;

-----------------------   Y2Ht   procedure---------------------}
 function Y2Ht (yin : extended) : integer;
   var
     tempyout : extended;
     yout : integer;
 begin
   with rlocusdata do
     begin
       if (ymin <= yin) and (yin <= ymax) then
         begin
           tempyout := topmar + plotht * (ymax - yin) / (ymax - ymin);
           yout := Num2Integer(tempyout)
         end   { if}
       else
```

```
        yout := 512;      { put it outside of plot }
      end;      { with }
    Y2Ht := yout;
  end;

{----------------------CrossPoint   procedure-----------------}
  procedure CrossPoint (xval, yval : extended);
    var
      ht, wd : integer;
  begin
    wd := X2Wd(xval);
    ht := Y2Ht(yval);
    if (wd <> 512) and (ht <> 512) then
      begin
        DrawLine(wd - 1, ht, wd + 1, ht);
        DrawLine(wd, ht - 1, wd, ht + 1);
      end;
  end;

{----------------------WriteNum   procedure-----------------}
  procedure WriteNum (numout : extended);
    var
      multiplier : extended;
      midmove, places, counter : integer;
  begin
    if (abs(numout) < 1e-7) then
      WriteDraw(0.0 : 3 : 1)
    else if (abs(numout) < 1e-3) or (abs(numout) > 1e3) then
      begin
        WriteDraw(numout);
      end
    else
      begin
        multiplier := 0.1;
        places := 3;
        while abs(numout) >= multiplier do
          begin
            multiplier := multiplier * 10;
            places := places - 1;
          end;
        if places <= 0 then
          places := 1;
        WriteDraw(numout : 3 : places);
      end;
  end;

{-----------------------  DrawHorizLine  procedure--------------------}
  procedure DrawHorizLine (yval : extended);
    var
      ht : integer;
  begin
    ht := Y2Ht(yval);
    DrawLine(ltmar, ht, ltmar + plotwd - 2, ht);
```

```
 end;

{------------------------ DrawVertLine procedure----------------------}
 procedure DrawVertLine (xval : extended);
  var
   wd : integer;
 begin
  wd := X2Wd(xval);
  DrawLine(wd, topmar, wd, topmar + plotht - 2);
 end;

{------------------------ InitRLocus procedure----------------------}
 procedure InitRLocus;
 begin
  DP := GetNewDialog(rlocusid, nil, pointer(-1));
  with rlocusdata do
   begin
    SetDData(DP, 10, Real2Str(mingain, true));
    SetDData(DP, 11, Real2Str(maxgain, true));
    SetDData(DP, 12, Int2Str(points));
    SetDData(DP, 18, Real2Str(xmin, true));
    SetDData(DP, 19, Real2Str(xmax, true));
    SetDData(DP, 20, Real2Str(ymin, true));
    SetDData(DP, 21, Real2Str(ymax, true));
    if linear then              { set point interval radio buttons }
     CheckDItem(DP, 6)
    else
     CheckDItem(DP, 7);
    if (simptype) and (sysgroupH^^.backbks <> 0) then          { set loop path radio button }
     CheckDItem(DP, 8)
    else
     CheckDItem(DP, 9);
    if (simptype) and (sysgroupH^^.backbks = 0) then          { set loop path radio button }
     begin
      Basic1Alert('There are no feedback blocks in the system so the default loop path has been changed
    to "Closed Loop" for this plot', 1);
      simptype := false;
     end;
   end;  { with rlocusdata}
  SelIText(DP, 10, 0, 255);  { select mingain data box  }
 end;

{------------------------ GoodRLocusDataEntered function----------------------}
 function GoodRLocusDataEntered : boolean;
  var
   tempmingain, tempmaxgain, tempxmin, tempxmax, tempymin, tempymax : extended;
   temppoints : integer;
   firsterr, valid, datagood, xgood, ygood : boolean;
 begin
  xgood := true;
  ygood := true;
  datagood := true;
  firsterr := true;
```

```
valid := GetCheckReal(DP, 10, tempmingain);     { check min gain }
if not valid or (tempmingain < 0) or (tempmingain > 1e7) then
  begin
    datagood := false;
    FrameDataError(firsterr, 10);
  end;
valid := GetCheckReal(DP, 11, tempmaxgain);     { check max gain }
if (not valid) or (tempmaxgain > 1e7) or (tempmaxgain <= 0) or (firsterr and (tempmaxgain <=
    tempmingain)) then
  begin
    datagood := false;
    FrameDataError(firsterr, 11);
  end;
GetCheckInt(DP, 12, temppoints, valid);
if not valid or (temppoints < 1) then
  begin
    datagood := false;
    FrameDataError(firsterr, 12);
  end;
valid := GetCheckReal(DP, 18, tempxmin);      { check xmin }
if not valid or (tempxmin < -1e7) or (tempxmin > 1e7) then
  begin
    datagood := false;
    xgood := false;
    FrameDataError(firsterr, 18);
  end;
valid := GetCheckReal(DP, 19, tempxmax);      { check xmax }
if not valid or (tempxmax < -1e7) or (tempxmax > 1e7) or (xgood and (tempxmax <= tempxmin))
    then
  begin
    datagood := false;
    FrameDataError(firsterr, 19);
  end;
valid := GetCheckReal(DP, 20, tempymin);      { check ymin }
if not valid or (tempymin < -1e7) or (tempymin > 1e7) then
  begin
    datagood := false;
    ygood := false;
    FrameDataError(firsterr, 20);
  end;
valid := GetCheckReal(DP, 21, tempymax);      { check ymax }
if not valid or (tempymax < -1e7) or (tempymax > 1e7) or (ygood and (tempymax <= tempymin))
    then
  begin
    datagood := false;
    FrameDataError(firsterr, 21);
  end;
GoodRlocusDataEntered := datagood;
if datagood then
  with rlocusdata do
    begin     { save good data }
      mingain := tempmingain;
      maxgain := tempmaxgain;
```

```
      points := temppoints;
      xmin := tempxmin;
      xmax := tempxmax;
      ymin := tempymin;
      ymax := tempymax;
      doit := true;
    end;
end;

{----------------------  GetRLocusData  procedure---------------------}
procedure GetRLocusData;
  var
    doagain, templinear, tempsimptype : boolean;
begin
  ClearAllWindows;
  InitRLocus;
  with rlocusdata do
    begin        { set temp buttons }
      templinear := linear;
      tempsimptype := simptype;
      doit := false;
    end;         { with rlocus data }
  doagain := true;      { set flag }
  while doagain do
    begin
      FrameDItem(DP, 1);
      ModalDialog(nil, itemnum);
      if (itemnum = 6) and (not templinear) then       { linear button }
        begin
          CheckDItem(DP, 6);
          CheckDItem(DP, 7);
          templinear := true;
        end;              { end linear button }
      if (itemnum = 7) and templinear then       { log button }
        begin
          CheckDItem(DP, 6);
          CheckDItem(DP, 7);
          templinear := false;
        end;           { end log button}
      if (itemnum = 8) and not tempsimptype and (sysgroupH^^.backbks <> 0) then { Geq button}
        begin
          CheckDItem(DP, 8);
          CheckDItem(DP, 9);
          tempsimptype := true;
        end;              { end Geq button }
      if (itemnum = 9) and tempsimptype then       { close loop button }
        begin
          CheckDItem(DP, 8);
          CheckDItem(DP, 9);
          tempsimptype := false;
        end;              { end closed loop button }
      if itemnum = 2 then          { cancel  }
        doagain := false;
```

```
      if itemnum = 1 then      { selected OK }
        begin
          doagain := not GoodRLocusDataEntered;
          if not doagain then
            begin     { save button data }
              rlocusdata.linear := templinear;
              rlocusdata.simptype := tempsimptype;
            end;
        end;{ ok }
    end;{ while }
  DisposDialog(DP);
end;

{------------------------  GetStep  procedure---------------------}
procedure GetStep (var xstep : extended;
                   var ystep : extended);
  var
    xspread, yspread : extended;
    newmin, newmax : longint;
begin
  with rlocusdata do
    begin
      xspread := xmax - xmin;
      yspread := ymax - ymin;
      if (xmax - xmin >= 10) then          { get x plot intervals}
        begin
          SetRound(downward);          { big spread}
          newmin := Num2Longint(xmin);
          SetRound(upward);
          newmax := Num2Longint(xmax);
          SetRound(tonearest);
          xmin := newmin;
          xmax := newmax;
          xstep := FindSep(newmax, newmin, 5);
        end
      else
        xstep := FindRealSep(xspread, 7);
      if (ymax - ymin >= 10) then          { get y plot intervals}
        begin
          SetRound(downward);          { big spread}
          newmin := Num2Longint(ymin);
          SetRound(upward);
          newmax := Num2Longint(ymax);
          SetRound(tonearest);
          ymin := newmin;
          ymax := newmax;
          ystep := FindSep(newmax, newmin, 8);
        end
      else
        ystep := FindRealSep(yspread, 8);
    end;      { with rlocus data }
end;
```

```
{----------------------- DrawBasicPlot procedure---------------------}
procedure DrawBasicPlot;
  const
    vertlabelsep = 15;
    horzlabelsep = 5;
  var
    xstep, ystep, linevalue : extended;
begin
  with rlocusdata do
    begin
      GetStep(xstep, ystep);
      PenSize(2, 2);
      FrameRect(plotrect);
      PenSize(1, 1);
      MoveTo(ltmar - 10, topmar + plotht + vertlabelsep - 2);      { x labels }
      WriteNum(xmin);
      MoveTo(ltmar + plotwd - 10, topmar + plotht + vertlabelsep - 2);
      WriteNum(xmax);
      linevalue := xmin + xstep;
      while linevalue < (xmax - xstep / 2) do
        begin
          DrawVertLine(linevalue);
          Move(-10, vertlabelsep);
          WriteNum(linevalue);
          linevalue := linevalue + xstep;
        end;  { while  }
      MoveTo(ltmar + plotwd + horzlabelsep - 2, topmar + plotht); { y labels }
      WriteNum(ymin);
      MoveTo(ltmar + plotwd + horzlabelsep - 2, topmar);
      WriteNum(ymax);
      linevalue := ymin + ystep;
      while linevalue < (ymax - ystep / 2) do
        begin
          DrawHorizLine(linevalue);
          Move(horzlabelsep, 0);
          WriteNum(linevalue);
          linevalue := linevalue + ystep;
        end;  { while  }
    end;    { with rlocusdata}
  PenSize(2, 2);
  DrawHorizLine(0);
  DrawVertLine(0);
  PenSize(1, 1);
end;

----------------------- GetGain procedure---------------------}
function NextGain : extended;
  var
    gainout, stepsize, logmin : extended;
begin
  with rlocusdata do
    begin
      if linear then
```

```
        begin        { linear }
          stepsize := (maxgain - mingain) / points;
          gainout := mingain + pointno * stepsize;
        end
      else          { logarithmic }
        begin
          if (mingain < 1e-5) and (maxgain > 1e-1) then
            logmin := -5
          else
            logmin := log(mingain);
          stepsize := (log(maxgain) - logmin) / points;
          gainout := Ten2(logmin + pointno * stepsize);
        end;
      end;        { with }
    NextGain := gainout;
  end;

{------------------------  GetGeq  procedure---------------------}
  function GetGeq (gain : extended;
                   var polyc : polycoef) : boolean;
    var
      unusedblock, G, H : block;
      valid, addneg : boolean;
      simpno : integer;
      leftpoly, rightpoly, charpoly, tempeq : polycoef;
  begin
    leftpoly := unityblock.num;
    rightpoly := unityblock.num;
    charpoly := unityblock.num;
    tempeq := unityblock.num;
    if rlocusdata.simptype then
      simpno := 1
    else
      simpno := 4;
    valid := GeqGroup(sysgroupH^^, simpno, unusedblock, G, H);
    if sysgroupH^^.posFback then    { set feedback type}
      addneg := false
    else
      addneg := true;
    if valid then
      begin
        if not PolyMult(G.den, H.den, leftpoly) then   { get firstleft poly }
          valid := false;
        if valid and not PolyMult(G.num, H.num, rightpoly) then
          valid := false;
        if simpno = 1 then                  { include gain factor if Geq  }
          rightpoly.gain := rightpoly.gain * gain;
        if valid then
          charpoly := PolySum(leftpoly, addneg, rightpoly);
        if simpno = 4 then
          begin     { closed loop  }
            tempeq := charpoly;
            if valid and not PolyMult(G.num, H.den, rightpoly) then
```

```
          valid := false;
        rightpoly.gain := rightpoly.gain * gain;
        charpoly := PolySum(tempeq, true, rightpoly);
      end;      { simptype =4 }
    end;  { if valid }
  polyc := charpoly;
  GetGeq := valid;
end;

 {-----------------------  PlotPoints  procedure---------------------}
procedure PlotPoints (polyfin : polyfact);
  var
    counter : integer;
begin
  with polyfin do
    begin
      for counter := 1 to degree do
        CrossPoint(-fact[counter].realpart,  fact[counter].imagpart);
    end;    { with }
end;

{-----------------------  DataToGraph  procedure---------------------}
procedure DataToGraph;
  var
    gain : extended;
    polycin : polycoef;
    continue : boolean;
    polyf : polyfact;
begin
  SetCursor(watch);
  ClipRect(plotrect);
  pointno := 0;
  DP := GetNewDialog(calcpointid, nil, pointer(-1));
  SetDData(DP, 1, 'Calculating data points. Please be patient!');
  SetDData(DP, 2, Int2Str(rlocusdata.points));
  while pointno <= rlocusdata.points do
    begin
      gain := NextGain;
      continue := GetGeq(gain, polycin);
      if continue then
        begin
          RootFinder(polycin, polyf, false);
          PlotPoints(polyf);
        end;  { if continue}
      SetDData(DP, 2, Int2Str(rlocusdata.points - pointno));
      pointno := pointno + 1;
    end;          { while }
  DisposDialog(DP);
  ClipRect(clipsize);
end;

{---------------------RedrawPlot  procedure-----------------}
procedure RedrawPlot;
```

```
  begin
    if not (sysgroupH^^.fwdbks + sysgroupH^^.backbks > 0) then
      Basic1Alert('There are no blocks in the system.', 1)
    else
      begin
        if not (rlocusdata.layer > 0) then
          Basic1Alert('A Root Locus has not yet beed drawn.', 1)
        else
          begin
            ShowWindow(rootPtr);
            SelectWindow(rootPtr);
          end;
      end;
  end;

{------------------------  DoRLocusMenu  procedure----------------------}
  procedure DoRLocusMenu;
  begin
    TextSize(9);
    plotht := scrnht - topmar - botmar;
    plotwd := scrnwd - ltmar - rtmar;
    SetRect(plotrect, ltmar, topmar, ltmar + plotwd, topmar + plotht);
    SetRect(clipsize, 0, 0, 512, 323);
    itemnum := alert(rlocusalertid, nil);
    case itemnum of
      1 :      { select Redraw }
        begin
          RedrawPlot;
        end;
      2 :    { select New Plot}
        begin
          if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
            begin
              GetRLocusData;
              if rlocusdata.doit then
                if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
                  begin
                    ShowWindow(rootPtr);
                    SelectWindow(rootPtr);
                    TextFace([bold]);
                    ClipRect(clipsize);
                    rlocusPic := OpenPicture(clipsize);
                    DrawBasicPlot;
                    DataToGraph;
                    ClosePicture;
                    SetWPic(rootPtr, rlocusPic);
                    SetCursor(arrow);
                    rlocusdata.layer := 1;
                  end;
            end
          else
            Basic1Alert('There are no blocks in the system.', 1);
        end;
```

```
    3 :    { select Overlap }
      begin
        if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
          begin
            if rlocusdata.layer > 0 then
              begin
                rlocusdata.layer := rlocusdata.layer + 1;
                HideWindow(rootPtr);
                ShowWindow(rootPtr);
                SelectWindow(rootPtr);
                ClipRect(clipsize);
                rlocusPic := GetWPic(rootPtr);
                newPic := OpenPicture(clipsize);
                DrawPicture(rlocusPic, clipsize);
                pensize(1, 1);
                penpat(black);
                SetCursor(watch);
                case rlocusdata.layer of
                  2 :
                    penpat(dkgray);
                  3 :
                    penpat(gray);
                  otherwise
                    penpat(ltgray);
                end;
                DataToGraph;
                SetCursor(arrow);
                ClosePicture;
                PenNormal;
                SetWPic(rootPtr, newPic);
                penpat(black);
                ShowWindow(rootPtr);
                SelectWindow(rootPtr);
              end
            else
              Basic1Alert('This would be the first plot.', 1);
          end
        else
          Basic1Alert('There are no blocks in the system.', 1);
      end;
    otherwise    { select Cancel }
      ;
  end;{ case }
  TextSize(12);
end;

end. { unit }
```

```
unit RFinder;
Interface
 uses
   XTTypeDefs, Extender1, CADGlobals, NumberCrunch, sane, DoBlockMenu;
 procedure DoRFinderMenu;
 var
   polycin : polycoef;
   polyfout : polyfact;
   polyorder : integer;
   doit, doagain, goodint : boolean;
Implementation
 procedure DoRFinderMenu;
 begin
   editnewblock := true;
   saveit := true;
   doit := true;
   doagain := true;
   DP := GetNewDialog(rootid, nil, pointer(-1));
   SelIText(DP, 4, 0, 255);
   while doagain do
     begin
       FrameDItem(DP, 1);
       ModalDialog(nil, itemnum);
       If itemNum = 2 then  { cancel was selected }
         begin
           doit := false;
           doagain := false;
         end { if}
       else      { ok was selected}
         begin
           GetCheckInt(DP, 4, polyorder, goodint);
           If (goodint) and (0 < polyorder) and (polyorder < 11) then{ the input no. was good }
             begin
               doit := true;
               doagain := false;
               polycin.degree := polyorder;
             end     { good input }
           else  { the number was not good}
             begin
               FrameDItem(DP, 4);
               SelIText(DP, 4, 0, 255);
               doagain := true;
             end;  { else not good number}
         end;{else OK selected}
     end;{while}
   DisposDialog(DP);
   If doit then
     begin
       LoadPolyCoef('RootFinder', polycin);  { load the poly }
       If saveit then
         begin
           SetCursor(watch);
           RootFinder(polycin, polyfout, true);   { solve the roots }
```

```
        SetCursor(arrow);
        DP := GetNewDialog(polyfactid, nil, pointer(-1));
        FrameDItem(DP, 1);
        OldFactDataOut(polyfout);
        ModalDialog(nil, itemnum);
        DisposDialog(DP);
      end;
    end;
  end;
end.
```

```
unit Time;
interface
  uses
    XTTypeDefs, Extender1, CADGlobals, SANE, NumberCrunch, Extend2Stuff;
  procedure DoTimeMenu;
implementation
  const
    timestepnumber = 1000;
    botmar = 35;
    topmar = 58;
    ltmar = 50;
    rtmar = 20;
    scrnht = 285;
    scrnwd = 493;
    timealertid = 29493;
    stepid = 13173;      { input type dialog box id's  }
    rampid = 8298;
    impulseid = 378;
    sineid = 19775;
  type
    matrix = array[1..20, 1..20] of extended;
    vector = array[1..20] of extended;
  var
    effgain : extended;        { block gain with den normalized  }
    tempblock : block;
    alertresponse, plotwd, plotht, oldx, oldy, poles, zeros : integer;
    Psi, Phi, A, Atemp : matrix;
    C, Xold, Xnew, Gamma : vector;
    WD : WData;
    timeclipsize, plotrect : rect;
    LRC, ULC : point;
    plotmaxmag, plotminmag, plotmagstep : extended;
    plotmaxtime, plottimestep, timeinterval, delt : extended; { timeinterval = time }
                              { between plotted points }
                             { delt is time between }
    timePic, newPic : PicHandle;              { state calculations  }
{------------------------------Mag2Ht    function------------------}
{ input a magnitude of the plot and an integer value of the height in }
{ pixels is returned.                        }
  function Mag2Ht (mag : extended) : integer;
    var
      tempht : integer;
  begin
    Mag2Ht := topmar + Num2Integer((plotmaxmag - mag) * plotht / (plotmaxmag - plotminmag));
  end;

{------------------------------Time2Wd    function------------------}
{ input a magnitude of the plot and an integer value of the height in }
{ pixels is returned.                        }
  function Time2Wd (time : extended) : integer;
    var
      tempwd : integer;
  begin
```

```
  Time2Wd := ltmar + Num2Integer(time * plotwd / plotmaxtime);
 end;

{------------------------------MatrixMult    function------------------}
 function MatrixMult (matrix1, matrix2 : matrix;
              order : integer) : matrix;
  var
   rowcount, colcount, sumcount : integer;
   tempmatrix : matrix;
 begin
  for rowcount := 1 to order do
   for colcount := 1 to order do
    tempmatrix[rowcount, colcount] := 0;
  for rowcount := 1 to order do
   for colcount := 1 to order do
    for sumcount := 1 to order do
     tempmatrix[rowcount, colcount] := tempmatrix[rowcount, colcount] + matrix1[rowcount,
     sumcount] * matrix2[sumcount, colcount];
  MatrixMult := tempmatrix;
 end;

{------------------------------ScalarMatrixMult    function------------------}
 function ScalarMatrixMult (matrixin : matrix;
              scalar : extended;
              order : integer) : matrix;
  var
   tempmatrix : matrix;
   rowcount, colcount : integer;
 begin
  for rowcount := 1 to order do
   for colcount := 1 to order do
    tempmatrix[rowcount, colcount] := matrixin[rowcount, colcount] * scalar;
  ScalarMatrixMult := tempmatrix;
 end;

{------------------------------MatrixVectorMult    function------------------}
 function MatrixVectorMult (matrixin : matrix;
              vectorin : vector;
              order : integer) : vector;
  var
   tempvector : vector;
   rowcount, colcount : integer;
   tempsum : extended;
 begin
  for rowcount := 1 to order do
   begin
    tempsum := 0;
    for colcount := 1 to order do
     tempsum := tempsum + Matrixin[rowcount, colcount] * vectorin[colcount];
    tempvector[rowcount] := tempsum;
   end;
  MatrixVectorMult := tempvector;
 end;
```

```
{-----------------------GoodStepDataEntered   function-----------------}
  function GoodStepDataEntered : boolean;
    var
      tempinamp, tempplotamp, temptime : extended;
      firsterr, valid, datagood : boolean;
  begin
    datagood := true;
    firsterr := true;
    valid := GetCheckReal(DP, 6, tempinamp);     { check input amp}
    if not valid or (tempinamp <= 0) or (tempinamp > 1e7) then
      begin
        datagood := false;
        FrameDataError(firsterr, 6);
      end;
    valid := GetCheckReal(DP, 7, tempplotamp);     { check plot amp }
{ make sure that if the input amp was good }
    if not valid or (tempplotamp > 1e7) or (tempplotamp <= 0) then
      begin
        datagood := false;
        FrameDataError(firsterr, 7);
      end;
    valid := GetCheckReal(DP, 8, temptime);      { check max time}
    if not valid or (temptime <= 0) or (temptime > 100) then
      begin
        datagood := false;
        FrameDataError(firsterr, 8);
      end;
    GoodStepDataEntered := datagood;
    if datagood then
      begin
        with timedata do
          begin
            amp := tempinamp;
            maxy := tempplotamp;
            maxtime := temptime;
          end;
      end;
  end;

{-----------------------------GetStepData   procedure-----------------}
  procedure GetStepData;
    var
      tempbutton, doagain : boolean;
      itemnum : integer;
  begin
    doagain := true;
    DP := GetNewDialog(stepid, nil, pointer(-1));
    SetDData(DP, 6, Real2Str(timedata.amp, true));
    SetDData(DP, 7, Real2Str(timedata.maxy, true));
    SetDData(DP, 8, Real2Str(timedata.maxtime, true));
    SelIText(DP, 6, 0, 255);
    if timedata.zerobottom then
```

```
    CheckDItem(DP, 10)
  else
    CheckDItem(DP, 11);
  tempbutton := timedata.zerobottom;
  while doagain do
    begin
      FrameDItem(DP, 1);
      ModalDialog(nil, itemNum);
      if itemNum = 2 then       { selected cancel }
        begin
          timedata.doit := false;
          doagain := false;
        end
      else if itemNum = 10 then  { checked bottom button }
        begin
          if not tempbutton then
            begin
              CheckDItem(DP, 10);
              CheckDItem(DP, 11);
            end;
          tempbutton := true;
        end
      else if itemNum = 11 then  { checked center button }
        begin
          if tempbutton then
            begin
              CheckDItem(DP, 10);
              CheckDItem(DP, 11);
            end;
          tempbutton := false;
        end
      else                  { selected OK   }
        begin
          doagain := not GoodStepDataEntered;
          if not doagain then
            timedata.zerobottom := tempbutton;
        end;
    end;
  DisposDialog(DP);
end;

{------------------------GoodRampDataEntered   function-----------------}
function GoodRampDataEntered : boolean;
  var
    tempslope, tempdcoff, tempplotamp, temptime : extended;
    firsterr, valid, datagood : boolean;
begin
  datagood := true;
  firsterr := true;
  valid := GetCheckReal(DP, 7, tempslope);     { check slope }
  if not valid or (abs(tempslope) > 1e7) then
    begin
      datagood := false;
```

```
      FrameDataError(firsterr, 7);
    end;
  valid := GetCheckReal(DP, 8, tempdcoff);     { check dc offset }
  if not valid or (abs(tempdcoff) > 1e7) then
    begin
      datagood := false;
      FrameDataError(firsterr, 8);
    end;
  valid := GetCheckReal(DP, 9, tempplotamp);    { check plot amp }
  if not valid or (tempplotamp <= 0) or (tempplotamp > 1e7) then
    begin
      datagood := false;
      FrameDataError(firsterr, 9);
    end;
  valid := GetCheckReal(DP, 10, temptime);    { check max time}
  if not valid or (temptime <= 0) or (temptime > 100) then
    begin
      datagood := false;
      FrameDataError(firsterr, 10);
    end;
  GoodRampDataEntered := datagood;
  if datagood then
    begin
      with timedata do
        begin
          slope := tempslope;
          dcoff := tempdcoff;
          maxy := tempplotamp;
          maxtime := temptime;
        end;
    end;
end;

{-----------------------------GetRampData   procedure-----------------}
procedure GetRampData;
  var
    tempbutton, doagain : boolean;
    itemnum : integer;
begin
  doagain := true;
  DP := GetNewDialog(rampid, nil, pointer(-1));
  SetDData(DP, 7, Real2Str(timedata.slope, true));
  SetDData(DP, 8, Real2Str(timedata.dcoff, true));
  SetDData(DP, 9, Real2Str(timedata.maxy, true));
  SetDData(DP, 10, Real2Str(timedata.maxtime, true));
  SelIText(DP, 7, 0, 255);
  if timedata.zerobottom then
    CheckDItem(DP, 12)
  else
    CheckDItem(DP, 13);
  tempbutton := timedata.zerobottom;
  while doagain do
    begin
```

```
      FrameDItem(DP, 1);
      ModalDialog(nil, itemNum);
      if itemNum = 2 then      { selected cancel }
        begin
          timedata.doit := false;
          doagain := false;
        end
      else if itemNum = 12 then  { checked bottom button }
        begin
          if not tempbutton then
            begin
              CheckDItem(DP, 12);
              CheckDItem(DP, 13);
            end;
          tempbutton := true;
        end
      else if itemNum = 13 then  { checked center button }
        begin
          if tempbutton then
            begin
              CheckDItem(DP, 12);
              CheckDItem(DP, 13);
            end;
          tempbutton := false;
        end
      else                { selected OK  }
        begin
          doagain := not GoodRampDataEntered;
          if not doagain then
            timedata.zerobottom := tempbutton;
        end;
    end;
  DisposDialog(DP);
end;

{----------------------GoodImpulseDataEntered   function-----------------}
  function GoodImpulseDataEntered : boolean;
    var
      tempinamp, tempplotamp, temptime : extended;
      firsterr, valid, datagood : boolean;
  begin
    datagood := true;
    firsterr := true;
    valid := GetCheckReal(DP, 6, tempinamp);    { check input amp}
    if not valid or (tempinamp <= 0) or (tempinamp > 1e7) then
      begin
        datagood := false;
        FrameDataError(firsterr, 6);
      end;
    valid := GetCheckReal(DP, 7, tempplotamp);    { check plot amp }
    if not valid or (tempplotamp <= 0) or (tempplotamp > 1e7) then
      begin
        datagood := false;
```

```
      FrameDataError(firsterr, 7);
    end;
  valid := GetCheckReal(DP, 8, temptime);      { check max time}
  If not valid or (temptime <= 0) or (temptime > 100) then
    begin
      datagood := false;
      FrameDataError(firsterr, 8);
    end;
  GoodImpulseDataEntered := datagood;
  If datagood then
    begin
      with timedata do
        begin
          impamp := tempinamp;
          maxy := tempplotamp;
          maxtime := temptime;
        end;
    end;
end;

{-----------------------------GetImpulseData   procedure-----------------}
procedure GetImpulseData;
  var
    temptime, displayedtime : extended;
    tempbutton, tempautoimpamp, doagain, valid, goodtimedisplayed : boolean;
    itemnum : integer;
begin
  doagain := true;
  DP := GetNewDialog(Impulseid, nil, pointer(-1));
  SetDData(DP, 6, Real2Str(timedata.impamp, true));
  If timedata.autoimpamp then                  { if auto imp amp set}
    SetDData(DP, 6, Real2Str(timestepnumber / timedata.maxtime, true));
  SetDData(DP, 7, Real2Str(timedata.maxy, true));
  SetDData(DP, 8, Real2Str(timedata.maxtime, true));
  SelIText(DP, 6, 0, 255);
  If timedata.zerobottom then     { set 'zero at bottom' radio button }
    CheckDItem(DP, 10)
  else                    { set 'zero at center' radio button }
    CheckDItem(DP, 11);
  If timedata.autoimpamp then       { set auto unit impulse amp check box }
    CheckDItem(DP, 12);
  tempautoimpamp := timedata.autoimpamp;       { set temp variables }
  tempbutton := timedata.zerobottom;
  temptime := timedata.maxtime;
  while doagain do
    begin
      FrameDItem(DP, 1);
      ModalDialog(nil, itemNum);
      If itemNum = 2 then       { selected cancel }
        begin
          timedata.doit := false;
          doagain := false;
        end
```

```
      else if itemNum = 12 then
        begin
          CheckDItem(DP, 12);
          tempautoimpamp := not tempautoimpamp;
          goodtimedisplayed := GetCheckReal(DP, 8, displayedtime);
          if tempautoimpamp and goodtimedisplayed then
            SetDData(DP, 6, Real2Str(timestepnumber / displayedtime, true));
        end
      else if (itemnum = 8) then      { if time box 'touched' }
        begin
          if (tempautoimpamp) then
            begin
              valid := GetCheckReal(DP, 8, temptime);      { check time }
              if valid and (temptime > 0) and (temptime <= 100) then
                begin
                  SetDData(DP, 6, Real2Str(timestepnumber / temptime, true));
                end;
            end;
          doagain := true;
        end
      else if (itemNum = 10) and not tempbutton then   { checked bottom button }
        begin
          CheckDItem(DP, 10);
          CheckDItem(DP, 11);
          tempbutton := true;
        end
      else if (itemNum = 11) and tempbutton then    { checked center button }
        begin
          CheckDItem(DP, 10);
          CheckDItem(DP, 11);
          tempbutton := false;
        end
      else                { selected OK   }
        begin
          doagain := not GoodImpulseDataEntered;
          if not doagain then
            begin
              timedata.autoimpamp := tempautoimpamp;
              timedata.zerobottom := tempbutton;
            end;
        end;
    end;
  DisposDialog(DP);
end;

{-----------------------GoodSineDataEntered   function------------------}
function GoodSineDataEntered : boolean;
  var
    tempinamp, tempfreq, tempplotamp, temptime : extended;
    firsterr, valid, datagood : boolean;
begin
  datagood := true;
  firsterr := true;
```

```
    valid := GetCheckReal(DP, 7, tempinamp);     { check input amp}
    if not valid or (tempinamp <= 0) or (tempinamp > 1e7) then
      begin
        datagood := false;
        FrameDataError(firsterr,  7);
      end;
    valid := GetCheckReal(DP, 8, tempfreq);     { check freq  }
    if not valid or (tempfreq <= 0) or (tempfreq > 1e3) then
      begin
        datagood := false;
        FrameDataError(firsterr,  8);
      end;
    valid := GetCheckReal(DP, 9, tempplotamp);     { check plot amp }
    if not valid or (tempplotamp <= 0) or (tempplotamp > 1e7) then
      begin
        datagood := false;
        FrameDataError(firsterr,  9);
      end;
    valid := GetCheckReal(DP, 10, temptime);     { check max time}
    if not valid or (temptime <= 0) or (temptime > 100) then
      begin
        datagood := false;
        FrameDataError(firsterr,  10);
      end;
    GoodSineDataEntered := datagood;
    if datagood then
      begin
        with timedata do
          begin
            amp := tempinamp;
            freq := tempfreq;
            maxy := tempplotamp;
            maxtime := temptime;
          end;
      end;
  end;

{-----------------------------GetSineData    procedure------------------}
  procedure GetSineData;
    var
      doagain : boolean;
      itemnum : integer;
  begin
    doagain := true;
    DP := GetNewDialog(sineid, nil, pointer(-1));
    SetDData(DP, 7, Real2Str(timedata.amp, true));
    SetDData(DP, 8, Real2Str(timedata.freq, true));
    SetDData(DP, 9, Real2Str(timedata.maxy, true));
    SetDData(DP, 10, Real2Str(timedata.maxtime, true));
    SelIText(DP, 7, 0, 255);
    while doagain do
      begin
        FrameDItem(DP, 1);
```

```
      ModalDialog(nil, itemNum);
      If itemNum = 2 then      { selected cancel }
        begin
          timedata.doit := false;
          doagain := false;
        end
      else                { selected OK   }
        begin
          doagain := not GoodSineDataEntered;
        end;
    end;
  DisposDialog(DP);
end;

{--------------CalculateMatrixAndVector   procedure-----------}
 procedure CalculateMatrixAndVector;
  var
    i, j, m, n : integer;
    factorial, T1, rowsum, maxrowsum, oldmaxrowsum : extended;
    finished : boolean;
 begin
  tempblock := sysblockH^^;      {   A MATRIX   }
  polynorm(tempblock.den);
  effgain := tempblock.num.gain / tempblock.den.gain;
  with tempblock do
    begin
      poles := den.degree;
      zeros := num.degree;
      for i := 1 to poles - 1 do
        for j := 1 to poles do
          If j = i + 1 then
            A[i, j] := 1     { fill with 0's and 1's  }
          else
            A[i, j] := 0;
      for j := 1 to poles do
        A[poles, j] := -den.coef[j];   { load last row with - den coefs}
    {   C MATRIX   }
      for i := 1 to poles do
        begin
          If i > zeros + 1 then
            C[i] := 0.0
          else
            C[i] := num.coef[i] * effgain;
          If zeros = poles then
            C[i] := C[i] + effgain * num.coef[zeros + 1] * A[poles, i];
        end;{ for i = 1 to poles }
    end;{ with tempblock }
    {   Psi and Atemp    }
  Atemp := A;
  Psi := A;
  Psi := ScalarMatrixMult(Psi, delt / 2, poles);
  for i := 1 to poles do
    Psi[i, i] := Psi[i, i] + 1.0;
```

```
    factorial := 2;
    T1 := delt;
    oldmaxrowsum := 0.0;
    repeat
      factorial := factorial * (poles + 1);
      T1 := T1 * delt;
      Phi := MatrixMult(A, Atemp, poles);
      Atemp := Phi;
      Phi := ScalarMatrixMult(Phi, (T1 / factorial), poles);
      for j := 1 to poles do
        for m := 1 to poles do
          Psi[j, m] := Psi[j, m] + phi[j, m];
      maxrowsum := 0.0;
      for j := 1 to poles do
        begin
          rowsum := 0.0;
          for m := 1 to poles do
            rowsum := rowsum + Psi[j, m];
          if rowsum > maxrowsum then
            maxrowsum := rowsum;
        end;
      if poles > 0 then
        begin
          if (abs(maxrowsum - oldmaxrowsum) / maxrowsum) < 0.001 then
            finished := false
          else
            finished := true;
        end
      else
        finished := true;
      oldmaxrowsum := maxrowsum;
    until finished;
    Psi := ScalarMatrixMult(Psi, delt, poles);
      { PHI MATRIX  }
    Phi := MatrixMult(A, Psi, poles);
    for i := 1 to poles do
      Phi[i, i] := Phi[i, i] + 1.0;
      { GAMMA VECTOR   }
    for i := 1 to poles do
      Gamma[i] := Psi[i, poles];
  end;

{-------------------DrawTimeLine   procedure-----------------}
  procedure DrawTimeLine (time, mag : extended);
    var
      newx, newy : integer;
  begin
    newx := Time2Wd(time);
    newy := Mag2Ht(mag);
    DrawLine(oldx, oldy, newx, newy);
    oldx := newx;
    oldy := newy;
  end;
```

```
{------------------CalculatePlotPoints  procedure-----------------}
 procedure CalculatePlotPoints;
  var
   i, j, n : integer;
   lasttimeout, magout, plottime, Uinput : extended;
   xold, xnew : vector;
 begin
  DP := GetNewDialog(calcpointid, nil, pointer(-1));
  SetDData(DP, 1, 'Calculating data points. Please be patient!');
  SetDData(DP, 2, Int2Str(timestepnumber));
  plottime := 0.0;
  lasttimeout := plottime;
  for i := 1 to poles do     { initialize states}
   xold[i] := 0.0;
  for n := 1 to timestepnumber do
   begin
    SetDData(DP, 2, Int2Str(timestepnumber - n));
    case timedata.inputtype of
     1 :    { step }
      Uinput := timedata.amp;
     2 :    { ramp}
      Uinput := timedata.dcoff + plottime * timedata.slope;
     3 :    { impulse }
      if plottime = 0.0 then
       Uinput := timedata.impamp
      else
       Uinput := 0.0;
     4 :    { sine wave }
      Uinput := timedata.amp * sin(timedata.freq * plottime);
    end;{ case }
    xnew := MatrixVectorMult(Phi, xold, poles);
    for i := 1 to poles do
     xnew[i] := xnew[i] + Gamma[i] * Uinput;
    magout := 0.0;
   { determine if time to display data, if so, do it }
    if plottime >= lasttimeout then
     begin
      lasttimeout := lasttimeout + timeinterval;
      for i := 1 to poles do
       magout := magout + C[i] * xnew[i];
      if poles = zeros then
       magout := magout + effgain * tempblock.num.coef[zeros + 1] * Uinput;
      if magout > plotmaxmag then    { avoid overflows }
       magout := 1.1 * plotmaxmag;
      if magout < plotminmag then
       magout := -1.1 * plotmaxmag;
      DrawTimeLine(plottime, magout);
     end;
    plottime := plottime + delt;
    xold := xnew;
   end;  {for n := 1 to timestepnumber do }
  DisposDialog(DP);
```

```
  end;

{-------------------DoDataPlot   procedure------------------}
  procedure DoDataPlot;
  begin
    CalculateMatrixAndVector;
    ClipRect(plotrect);
    pensize(2, 2);
    CalculatePlotPoints;
    ClipRect(timeclipsize);
  end;

{-------------------DoHorizGrid   procedure------------------}
  procedure DoHorizGrid;
    const
      labelmar = 3;
    var
      magtodraw : extended;
      pixelht, zeroht : integer;          { mag ht in pixels}
  begin
    pensize(1, 1);
    TextSize(9);
    MoveTo(labelmar, topmar + plotht);
    WriteIt(plotminmag, plotmagstep);
    MoveTo(labelmar, topmar);
    WriteIt(plotmaxmag, plotmagstep);
    magtodraw := plotminmag;
    while (magtodraw < plotmaxmag - plotmagstep) do
      begin
        magtodraw := magtodraw + plotmagstep;
        pixelht := Mag2Ht(magtodraw);
        DrawLine(ltmar, pixelht, ltmar + plotwd - 1, pixelht);
        MoveTo(labelmar, pixelht);
        WriteIt(magtodraw, plotmagstep);
      end;{ while }
    if (plotminmag < 0) and (0 < plotmaxmag) then
      begin
        zeroht := Mag2Ht(0);
        DrawLine(ltmar, zeroht, ltmar + plotwd - 1, zeroht);
      end;
  end;

{-------------------DoVertGrid   procedure------------------}
  procedure DoVertGrid;
    const
      labelmar = 9;
      labelshiftleft = 28;
    var
      timetodraw : extended;
      pixelwd, zeroht : integer;          { mag ht in pixels}
  begin
    pensize(1, 1);
    TextSize(9);
```

```
    MoveTo(ltmar - labelshiftleft, topmar + plotht + labelmar);
    WriteIt(0, plottimestep);
    MoveTo(ltmar + plotwd - labelshiftleft, topmar + plotht + labelmar);
    WriteIt(plotmaxtime, plottimestep);
    timetodraw := 0;
    while (timetodraw < plotmaxtime - plottimestep) do
      begin
        timetodraw := timetodraw + plottimestep;
        pixelwd := Time2Wd(timetodraw);
        DrawLine(pixelwd, topmar, pixelwd, topmar + plotht - 1);
        MoveTo(pixelwd - labelshiftleft, topmar + plotht + labelmar);
        WriteIt(timetodraw, plottimestep);
      end;{ while }
    MoveTo(ltmar + Num2Integer((plotwd - StringWidth('Time   (secs)')) / 2), scrnht - 11);
    DrawString('Time   (secs)');
  end;

{------------------DrawBasicPlot   procedure-----------------}
  procedure DrawBasicPlot;
  begin
    SetCursor(watch);
    ShowWindow(timePtr);
    SelectWindow(timePtr);
    ClipRect(timeclipsize);
    timePic := OpenPicture(timeclipsize);
    pensize(2, 2);
    FrameRect(plotrect);
    DoHorizGrid;
    DoVertGrid;
    DoDataPlot;
    ClipRect(timeclipsize);
    ClosePicture;
    SetWPic(timePtr, timePic);
    PenNormal;
    SetCursor(arrow);
  end;

{------------------CalculatePlotDimensions   procedure-----------------}
  procedure CalculatePlotDimensions;
    const
      timepoints = 3;  { no. of pixels between plotted time points. }
    var
      tempmagint, temptimeint : longint;   { for use with FindSep }
  begin
    with timedata do
      begin
        plotmaxtime := maxtime;
        timeinterval := plotmaxtime * timepoints / plotwd;
        delt := plotmaxtime / timestepnumber;
        plotmaxmag := maxy;
        If plotmaxmag < 10 then      { is the max plot mag less than ten }
          plotmagstep := FindRealSep(plotmaxmag, 7){ will adjust plotmaxmag to next hi int value }
        else
```

```
        begin
          SetRound(upward);
          tempmagint := Num2Longint(maxy);   { rounds up to next int value }
          SetRound(tonearest);
          plotmagstep := FindSep(tempmagint, 0, 7);
          plotmaxmag := tempmagint;
        end;
      if (not zerobottom) or (inputtype = 4) then   { if zero is center of the plot or input sine }
        begin
          plotminmag := -plotmaxmag;
          plotmagstep := 2 * plotmagstep;
        end
      else                          { if zero is the bottom of the plot }
        plotminmag := 0;
      if plotmaxtime < 10 then       { is the max plot time less than ten }
        plottimestep := FindRealSep(plotmaxtime, 8) { will adjust plotmaxtime to next hi int value }
      else
        begin
          SetRound(upward);
          temptimeint := Num2Longint(maxtime);   { rounds up to next int value }
          SetRound(tonearest);
          plottimestep := FindSep(temptimeint, 0, 10);
          plotmaxtime := temptimeint;
        end;
    end;{ with }
  oldx := Time2Wd(0);
  oldy := Mag2Ht(0);
end;

{-----------------------------DoTimeMenu   procedure------------------}
  procedure DoTimeMenu;
  begin
    pennormal;
    TextFace([bold]);
    timedata.doit := true;
    plotht := scrnht - topmar - botmar;
    plotwd := scrnwd - ltmar - rtmar;
    LRC.v := topmar + plotht;
    LRC.h := ltmar + plotwd;
    ULC.v := topmar;
    ULC.h := ltmar;
    plotrect.topleft := ULC;     { outline for plot }
    plotrect.botright := LRC;
    SetRect(timeclipsize, 0, 0, 512, 323);
    alertresponse := Alert(timealertid, nil);   { determine input/overlay}
    case alertresponse of
      1 :    { redraw plot }
        begin
          if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
            begin
              if timedata.layer > 0 then
                begin
                  ShowWindow(timePtr);
```

```
          SelectWindow(timePtr);
        end
      else
        Basic1Alert('A Time Response has not yet been plotted.', 1);
    end
  else
    Basic1Alert('There are no blocks in the system.', 1);
end;
2 :   { overlap plots }
begin
  if (sysgroupH^^.fwdbks + sysgroupH^^.backbks > 0) then
    begin
      if timedata.layer > 0 then
        begin
          CalculatePlotDimensions;
          timedata.layer := timedata.layer + 1;
          HideWindow(TimePtr);
          ShowWindow(timePtr);
          SelectWindow(timePtr);
          ClipRect(timeclipsize);
          timePic := GetWPic(timePtr);
          newPic := OpenPicture(timeclipsize);
          DrawPicture(timePic, timeclipsize);
          case timedata.layer of
            2 :
              penpat(dkgray);
            3 :
              penpat(gray);
            otherwise
              penpat(ltgray);
          end; { case }
          DoDataPlot;
          penpat(black);
          ClosePicture;
          SetWPic(timePtr, newPic);
          ShowWindow(timePtr);
          SelectWindow(timePtr);
        end
      else
        Basic1Alert('This would be the first plot.', 1);
    end
  else
    Basic1Alert('There are no blocks in the system.', 1);
end;
3 :   { step input }
begin
  if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
    begin
      timedata.inputtype := 1;
      ClearAllWindows;
      GetStepData;
      if timedata.doit then
        begin
```

```
            CalculatePlotDimensions;
            timedata.layer := 1;
            DrawBasicPlot;
          end;
      end
    else
      Basic1Alert('There are no blocks in the system.', 1);
  end;
4 :   { ramp input }
  begin
    if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
      begin
        timedata.inputtype := 2;
        ClearAllWindows;
        GetRampData;
        if timedata.doit then
          begin
            CalculatePlotDimensions;
            timedata.layer := 1;
            DrawBasicPlot;
          end;
      end
    else
      Basic1Alert('There are no blocks in the system.', 1);
  end;
5 :   { inpulse}
  begin
    if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
      begin
        timedata.inputtype := 3;
        ClearAllWindows;
        GetImpulseData;
        if timedata.doit then
          begin
            CalculatePlotDimensions;
            timedata.layer := 1;
            DrawBasicPlot;
          end;
      end
    else
      Basic1Alert('There are no blocks in the system.', 1);
  end;
6 :   { sinewave }
  begin
    if (sysgroupH^^.fwdbks + sysgroupH^^.backbks) > 0 then
      begin
        timedata.inputtype := 4;
        ClearAllWindows;
        GetSineData;
        if timedata.doit then
          begin
            CalculatePlotDimensions;
            timedata.layer := 1;
```

```
            DrawBasicPlot;
          end;
        end
      else
        Basic1Alert('There are no blocks in the system.', 1);
    end;
  7 :   { cancel }
    begin
      timedata.doit := false;
    end;
  otherwise
    ;
 end;{ case }
 TextSize(12);
end;

end. { unit }
```

```
unit simpgroup;
interface
  uses
    xttypedefs, extender1, CadGlobals, NumberCrunch, CADSetUp;
  procedure SimpSysGroup;

  function GeqGroup (grouptosimp : group;
                     simpID : integer;
                     var Gsimp : block;
                     var G : block;
                     var H : block) : boolean;
implementation

{------------------------------GeqGroup--------------------------}
  function GeqGroup;
       { (grouptosimp : group; simpID : integer; var Gsimp,G,H : block): boolean;}
    var
      temp1, temp2 : polycoef;
      counter : integer;
      GoodSimp : boolean;
      Geq : block;
  begin
    goodsimp := true;
    Gsimp := unityblock;
    G := unityblock;
    H := unityblock;
    Geq := unityblock;
    if (grouptosimp.backbks = 0) and (simpId <> 3) then      { initializes H to zero if no feedback }
      H.num.gain := 0;
    temp1 := unityblock.num;
    temp2 := unityblock.num;

    for counter := 1 to 5 do          { set up G and H blocks}
      if goodsimp then                { if all mults are good so far  }
        with grouptosimp.bksused[counter]^^ do
          if used then                { if this block is used}
            begin
              if forward then           { if in forward path, put in G  }
                begin
                  if not (polymult(G.num, num, G.num) and polymult(G.den, den, G.den)) then
                    goodsimp := false;
                end
              else                    { if in back path put in H }
                begin
                  if not (polymult(H.num, num, H.num) and polymult(H.den, den, H.den)) then
                    goodsimp := false;
                end
            end;  { G/H block set up }
    case simpID of
      2 :                         { forward  path }
        Geq := G;
      3 :                         { open loop  }
        if not (PolyMult(G.num, H.num, Geq.num) and PolyMult(G.den, H.den, Geq.den)) then
```

```
        goodsimp := false;
    otherwise                    { Geq or closed loop }
      begin
        if not (PolyMult(G.num, H.den, Geq.num)) then    { find Geq num }
          goodsimp := false;
        if not (PolyMult(G.den, H.den, temp1) and PolyMult(G.num, H.num, temp2)) then
                        { find two terms to add in Geq den }
          goodsimp := false;
        if goodsimp then
          Geq.den := PolySum(temp1, (not grouptosimp.posFback), temp2);        { find Geq den}
      end;
  end;  { case }
  if (simpID = 4) then              { if closed loop }
    Geq.den := PolySum(Geq.num, true, Geq.den);
  GeqGroup := goodsimp;
  if goodsimp then
    Gsimp := Geq
  else
    Basic1Alert('This group could not be simplified. Check that the order would not be above limits.', 2);
end;

{------------------------------SimpSysGroup--------------------------}
procedure SimpSysGroup;
  var
    newblockH : bksHdl;
    newtitle  : str255;
    done, makechange : boolean;
begin
  makechange := true;
  repeat
    done := true;
    DP := GetNewDialog(getstringid, nil, pointer(-1));
    SelIText(DP, 4, 0, 255);
    FrameDItem(DP, 1);
    ModalDialog(nil, itemNum);          { show the 'input name' dialog}
    GetDData(DP, 4, newtitle);          { get the new name }
    if itemNum = 2 then             { if cancel  }
      makechange := false
    else if length(newtitle) > 45 then     { if title is too long  }
      begin
        done := false;
        Basic1Alert('The title can not be longer than 45 characters.', 2)
      end;                  { if not too long, make the change}
    DisposDialog(DP);
  until done;
  if makechange then
    begin
      sysgroupH^^.maingrp := false;
      newblockH := sysblockH;       { set new block = old sysblock    }
      InitBks;                  { set and clear blocks in new sysgroup }
      newblockH^^.title := newtitle;
      sysgroupH^^.fwdbks := 1;      { only one block in new sysgroup }
      newblockH^^.fromgrpHdl := sysgroupH^^.ownHdl;
```

```
      sysgroupH^^.bksused[1] := newblockH;
      sysblockH^^.simpform := newblockH^^.simpform;
      sysblockH^^.num := newblockH^^.num;
      sysblockH^^.den := newblockH^^.den;
    end;
  end;{ DoSimplifyGroup }

end. { this module }
```

```
unit AddLabel;
interface
 uses
   XTTypeDefs, Extender1, CADGlobals, NumberCrunch, SANE, Extend2Stuff;
 procedure DoLabelMenu;
implementation
 const
   ltmar = 15;          { indent to text from left }
   initheight = 24;      { starting rect height   }
   addheight = 12;       { inc height for each line}
   borderspace = 2;       { space between border rect }
 var
   lefth, topv : integer;
   window2change : windowPtr;
   tempPic, labelPic, combinedPic : PicHandle;
   clipsize : rect;
   line1, line2, line3 : str255;
   lineno : integer;
   labelbox : rect;
   nostr : str255;
   dolabel : boolean;
   boxht, boxwd : integer;

{-----------------------GetLabelData   procedure-----------------}
 procedure GetLabelData;
   var
     newstrwidth, tempstrwidth : integer;
 begin
   TextFace([bold]);
   nostr := '';
   DP := GetNewDialog(labeldid, nil, pointer(-1));
   FrameDItem(DP, 1);
   SelIText(DP, 5, 0, 255);
   ModalDialog(nil, itemNum);
   if itemNum = 2 then
     dolabel := false
   else
     begin
       tempstrwidth := 0;    { init string width }
       dolabel := true;
       GetDData(DP, 5, line1);
   { get dialog box texts }
       GetDData(DP, 6, line2);
       GetDData(DP, 7, line3);
       if line1 <> nostr then   { if first line is used }
         begin
           lineno := 1;
           tempstrwidth := StringWidth(line1);
         end;
       if line2 <> nostr then   { if second line used }
         begin
           lineno := 2;
           newstrwidth := StringWidth(line2);
```

```
        if newstrwidth > tempstrwidth then  { find longest str }
          tempstrwidth := newstrwidth;
      end;
      if line3 <> nostr then   { if third line used }
        begin
          lineno := 3;
          newstrwidth := StringWidth(line3);
          if newstrwidth > tempstrwidth then  { find longest str }
            tempstrwidth := newstrwidth;
        end;
      boxwd := 2 * ltmar + tempstrwidth;       { set box width  }
      boxht := initheight + (lineno - 1) * addheight; { set box height  }
    end;  { OK button hit }
  DisposDialog(DP);
end;

{------------------------DrawLabel   procedure------------------}
  procedure DrawLabel;
    var
      temprect : rect;
  begin
    EraseRect(labelbox);
    Pensize(1, 1);
    FrameRect(labelbox);
    PenSize(2, 2);
    temprect := labelbox;
    InsetRect(temprect, 2, 2);
    FrameRect(temprect);
    MoveTo(labelbox.left + ltmar, labelbox.top + 17);
    DrawString(line1);
    MoveTo(labelbox.left + ltmar, labelbox.top + 17 + addheight);
    DrawString(line2);
    MoveTo(labelbox.left + ltmar, labelbox.top + 17 + 2 * addheight);
    DrawString(line3);
  end;

{------------------------DragBox   procedure------------------}
  procedure DragBox;
    var
      newpt, oldpt, currentpoint : point;
      hOffset, vOffset, counter : integer; {for comparing mouse pos }
      OldBox, MoveBox : rect;          { for moving icon box  }
  begin
    PenMode(PatXor);
    PenPat(gray);
    MoveBox := labelbox; { starting posit for move box   }
    OldBox := MoveBox;
    frameRect(OldBox);
    GetMouse(currentpoint.h, currentpoint.v);
    newpt := currentpoint;
    oldpt := newpt;
    while button do     { begin moving box algorithm   }
      begin
```

```
      GetMouse(newpt.h, newpt.v);
      If (abs(newpt.h - oldpt.h) > 1) or (abs(newpt.v - oldpt.v) > 1) then
        begin
          frameRect(OldBox);
          hOffset := newpt.h - oldpt.h;
          vOffset := newpt.v - oldpt.v;
          OffsetRect(MoveBox, hOffset, vOffset);
          FrameRect(MoveBox);
          oldpt := newpt;
          OldBox := MoveBox;
        end;
    end;
  FrameRect(MoveBox);{ erases last MoveBox    }
  labelbox := Oldbox;
end;

{------------------------DoLabelMenu   procedure------------------}
procedure DoLabelMenu;
begin
  window2change := FrontWindow;
  If window2change <> nil then
    begin
      GetLabelData;

      If dolabel then
        begin
          SetRect(clipsize, 0, 0, 512, 323);
          ClipRect(clipsize);
          tempPic := GetWPic(window2change);
          SetWPic(window2change, tempPic);
          UpdateWindow(window2change);
          ClipRect(clipsize);
          while not button do
            GetMouse(lefth, topv);
          SetRect(labelbox, lefth, topv, lefth + boxwd, topv + boxht);
          DragBox;
          GetMouse(lefth, topv);
          ShowWindow(window2change);
          SelectWindow(window2change);
          labelPic := OpenPicture(clipsize);
          PenNormal;
          DrawLabel;
          ClosePicture;
          DrawLabel;
          PenNormal;
          ignore := Alert(savelabelaid, nil);
          If ignore = 1 then
            begin
              combinedPic := AddPic(tempPic, labelPic);
              SetWPic(window2change, combinedPic);
            end
          else
            begin
```

```
            HideWindow(window2change);
            ShowWindow(window2change);
            SelectWindow(window2change);
          end;
      end;
    end
   else
    Basic1Alert('A window must be displayed in order to add a label.', 1);
 end;

end. { unit }
```

```
unit PrintWindow;
Interface
 uses
   XTTypeDefs, Extender1, CADGlobals, NumberCrunch, SANE, Extend2Stuff;
 procedure DoPrintMenu;
Implementation
 const
  { Printing Methods }
   bDraftLoop = 0;
   bSpoolLoop = 1;
   bUser1Loop = 2;
   bUser2Loop = 3;
  { Printers }
   bDevCItoh = 1;
   iDevCItoh = $0100;  {CItoh}
   bDevDaisy = 2;
   iDevDaisy = $0200;  {Daisy}
   bDevLaser = 3;
   iDevLaser = $0300;  {Laser}
  { PrCtlCall parameters }
   iPrBitsCtl = 4;
   lScreenBits = $00000000;
   lPaintBits = $00000001;
   lHiScreenBits = $00000010;
   lHiPaintBits = $00000011;
   iPrIOCtl = 5;
   iPrEvtCtl = 6;
   lPrEvtAll = $0002FFFD;
   lPrEvtTop = $0001FFFD;
   iPrDevCtl = 7;
   lPrReset = $00010000;
   lPrPageEnd = $00020000;
   lPrLineFeed = $00030000;
   lPrLFSixth = $0003FFFF;
   lPrLFEighth = $0003FFFE;
   iFMgrCtl = 8;
 { Result Codes }
   iMemFullErr = -108;
   iPrAbort = 128;
   iIOAbort = -27;
   iPrSavPFil = -1;
 { Miscellaneous }
   sPrDrvr = '.Print';
   iPrDrvrRef = -3;
   iPrPgFract = 120;
   iPrPgFst = 1;
   iPrPgMax = 9999;
   iPrRelease = 3;
   iPfMaxPgs = 128;
   pPrGlobals = $00000944;
 type
   TPRect = ^Rect;
   TPBitMap = ^BitMap;
```

```
TPrVars = record
    iPrErr : Integer;
    bDocLoop : SignedByte;
    bUser1 : SignedByte;
    lUser1 : LongInt;
    lUser2 : LongInt;
    lUser3 : LongInt;
  end;
TPPrVars = ^TPrVars;
TPrInfo = record
    iDev : Integer;
    iVRes : Integer;
    iHRes : Integer;
    rPage : Rect;
  end;
TPPrInfo = ^TPrInfo;
TFeed = (feedCut, feedFanfold, feedMechCut, feedOther);
TPrStl = record
    wDev : Integer;
    iPageV : Integer;
    iPageH : Integer;
    bPort : SignedByte;
    feed : TFeed;
  end;
TPPrStl = ^TPrStl;
TScan = (scanTB, scanBT, scanLR, scanRL);
TPrXInfo = record
    iRowBytes : Integer;
    iBandV : Integer;
    iBandH : Integer;
    iDevBytes : Integer;
    iBands : Integer;
    bPatScale : SignedByte;
    bULThick : SignedByte;
    bULOffset : SignedByte;
    bULShadow : SignedByte;
    scan : TScan;
    bXInfoX : SignedByte;
  end;
TPPrXInfo = ^TPrXInfo;
TPrJob = record
    iFstPage : Integer;
    iLstPage : Integer;
    iCopies : Integer;
    bJDocLoop : SignedByte;
    fFromUsr : Boolean;
    pIdleProc : ProcPtr;
    pFileName : StringPtr;
    iFileVol : Integer;
    bFileVers : SignedByte;
    bJobX : SignedByte;
  end;
TPPrJob = ^TPrJob;
```

```
  TPrint = record
     iPrVersion : Integer;
     PrInfo : TPrInfo;
     rPaper : Rect;
     PrStl : TPrStl;
     PrInfoPT : TPrInfo;
     PrXInfo : TPrXInfo;
     PrJob : TPrJob;
     PrintX : array[1..19] of Integer;
   end;
  TPPrint = ^TPrint;
  THPrint = ^TPPrint;
  TPrPort = record
     GPort : GrafPort;
     GProcs : QDProcs;
     lGParam1 : LongInt;
     lGParam2 : LongInt;
     lGParam3 : LongInt;
     lGParam4 : LongInt;
     fOurPtr : Boolean;
     fOurBits : Boolean;
   end;
  TPPrPort = ^TPrPort;
  TPrStatus = record
     iTotPages : Integer;
     iCurPage : Integer;
     iTotCopies : Integer;
     iCurCopy : Integer;
     iTotBands : Integer;
     iCurBand : Integer;
     fPgDirty : Boolean;
     fImaging : Boolean;
     hPrint : THPrint;
     pPrPort : TPPrPort;
     hPic : PicHandle;
   end;
  TPPrStatus = ^TPrStatus;
  TPfPgDir = record
     iPages : Integer;
     lPgPos : array[0..iPfMaxPgs] of LongInt;
   end;
  TPPfPgDir = ^TPfPgDir;
  THPfPgDir = ^TPPfPgDir;
  TPfHeader = record
     Print : TPrint;
     PfPgDir : TPfPgDir;
   end;
  TPPfHeader = ^TPfHeader;
  THPfHeader = ^TPPfHeader;
  TPrDlg = record
     Dlg : DialogRecord;
     pFltrProc : ProcPtr;
     pItemProc : ProcPtr;
```

```
      hPrintUsr : THPrint;
      fDoIt : Boolean;
      fDone : Boolean;
      lUser1 : LongInt;
      lUser2 : LongInt;
      lUser3 : LongInt;
      lUser4 : LongInt;
    { ...Plus more stuff needed by the particular printing dialog... }
    end;
  TPPrDlg = ^TPrDlg;
{ Initialization }
 procedure PrOpen;
 external;
 procedure PrClose;
 external;
{ Print Dialogs & Default }
 procedure PrintDefault (hPrint : THPrint);
 external;
 function PrValidate (hPrint : THPrint) : Boolean;
 external;
 function PrStlDialog (hPrint : THPrint) : Boolean;
 external;
 function PrJobDialog (hPrint : THPrint) : Boolean;
 external;
 procedure PrJobMerge (hPrintSrc, hPrintDst : THPrint);
 external;
{ Document printing procs: These spool a print file. }
 function PrOpenDoc (hPrint : THPrint;
              pPrPort : TPPrPort;
              pIOBuf : Ptr) : TPPrPort;
 external;
 procedure PrCloseDoc (pPrPort : TPPrPort);
 external;
 procedure PrOpenPage (pPrPort : TPPrPort;
              pPageFrame : TPRect);
 external;
 procedure PrClosePage (pPrPort : TPPrPort);
{ The "Printing Application" proc: Read and band the spooled PicFile. }
 external;
 procedure PrPicFile (hPrint : THPrint;
              pPrPort : TPPrPort;
              pIOBuf : Ptr;
              pDevBuf : Ptr;
              var PrStatus : TPrStatus);
{ Get/Set the current Print Error }
 external;
 function PrError : Integer;
 external;
 procedure PrSetError (iErr : Integer);
{ The .Print driver calls. }
 external;
 procedure PrDrvrOpen;
 external;
```

```
  procedure PrDrvrClose;
  external;
  procedure PrCtlCall (iWhichCtl : Integer;
                lParam1, lParam2, lParam3 : LongInt);
{ Semi private stuff }
  external;
  function PrStlInit (hPrint : THPrint) : TPPrDlg;
  external;
  function PrJobInit (hPrint : THPrint) : TPPrDlg;
  external;
  function PrDlgMain (hPrint : THPrint;
                pDlgInit : ProcPtr) : Boolean;
  external;
  procedure PrPurge;
  external;
  procedure PrNoPurge;
  external;
  function PrDrvrDCE : Handle;
  external;
  function PrDrvrVers : Integer;
  external;
  function PrintWPic (W : WindowPtr;
                hPrint : THPrint) : OSErr;
  external;
  function NewPrintHandle : THPrint;
    var
      hPrint : THPrint;
  begin
    PrOpen;                    { Be sure Printing Manager is open }
    hPrint := THPrint(NewHandle(SIZEOF(TPrint)));{ Allocate memory on heap }
    PrintDefault(hPrint);          { Set print record to default values }
    NewPrintHandle := hPrint;
  end;

{--------------------DoPrintMenu   procedure----------}
  procedure DoPrintMenu;
    var
      printH : THPrint;
      temperr : OSErr;
      doit : boolean;
      window2print : windowPtr;
  begin
    window2print := FrontWindow;
    if window2print <> nil then
      begin
        PrOpen;
        printH := NewPrintHandle;
        doit := PrStlDialog(printH);
        if doit then
          doit := PrJobDialog(printH);
        if doit then
          begin
            SetCursor(watch);
```

```
        temperr := PrintWPic(window2print, printH);
        SetCursor(arrow);
      end;
    PrClose;
  end
 else
  Basic1Alert('There must be a window displayed in order to print.', 1);
end;

end.
```

```
unit WindowTile;
Interface
  uses
    XTTypeDefs, XTTypeDefs2, XTDataIO, CADGlobals, CADSetUp, Extender1, NumberCrunch;
  procedure DoWindowMenu (item : integer);
  procedure SaveFile;
  procedure OpenFile (anoldfile : boolean);
Implementation
  var
    thedata : datalist;
  procedure TileFill (boundsRect : Rect;
                      theList : DataList;
                      animate : Boolean);
  external;
  procedure TileVertical (boundsRect : Rect;
                      theList : DataList;
                      animate : Boolean);
  external;
  procedure TileHorizontal (boundsRect : Rect;
                      theList : DataList;
                      animate : Boolean);
  external;
  procedure StackWindows (boundsRect : Rect;
                      theList : DataList;
                      animate : Boolean);
  external;

{----------------------------MoveBack   procedure-------------------}
  procedure MoveBack;
    var
      tempwindow : windowPtr;
  begin
    tempwindow := FrontWindow;
    if tempwindow = nil then
      Basic1Alert('No windows are displayed.', 1)
    else
      SendBehind(tempwindow, nil);
  end;

{----------------------------FillDataList   procedure-----------------}
  procedure FillDataList;
    var
      windowtoload, firstwindow : windowPtr;
      counter : integer;
      done : boolean;
  begin
    done := false;
    InitDataList(thedata);
    counter := -1;
    repeat
      windowtoload := FrontWindow;
      if windowtoload <> nil then
        begin
```

```
        counter := counter + 1;
        if counter = 0 then              { set first window flag }
          begin
            firstwindow := windowtoload;
            thedata.pItem[0] := Ptr(windowtoload);
            MoveBack;
          end
        else if windowtoload = firstwindow then          { check flag }
          done := true                   { started repeating }
        else
          begin
            thedata.pItem[counter] := Ptr(windowtoload);
            MoveBack;
          end;
      end
    else          { windowtoload = nil }
      done := true;
  until done;
  if counter = -1 then
    Basic1Alert('There are no plots displayed.', 1);
  thedata.numHandles := counter;
  thedata.numPointers := counter;
end;

{----------------------------ShowAll   procedure-------------------}
  procedure ShowAll (var plotexists : boolean);
  begin
    plotexists := false;
    if timedata.layer <> 0 then
      begin
        plotexists := true;
        ShowWindow(timePtr);
      end;
    if nyquistdata.layer <> 0 then
      begin
        plotexists := true;
        ShowWindow(nyqPtr);
      end;
    if rlocusdata.layer <> 0 then
      begin
        plotexists := true;
        ShowWindow(rootPtr);
      end;
    if bodedata.layer <> 0 then
      begin
        plotexists := true;
        ShowWindow(bodePtr);
      end;
  end;

{----------------------------CloseFront   procedure-------------------}
  procedure CloseFront;
    var
```

```
    tempwindow : windowPtr;
  begin
    tempwindow := FrontWindow;
    if tempwindow = nil then
      Basic1Alert('No windows are displayed.', 1)
    else
      HideWindow(tempwindow);
  end;

{----------------------------DoWindowMenu   procedure-------------------}
  procedure DoWindowMenu;
    var
      plotshown : boolean;
  begin
    if item < 5 then
      FillDataList;
    case item of
      1 :
        TileFill(screenBits.bounds, thedata, true);
      2 :
        TileVertical(screenBits.bounds, thedata, true);
      3 :
        TileHorizontal(screenBits.bounds, thedata, true);
      4 :
        StackWindows(screenBits.bounds, thedata, true);
      5 :
        begin
          ShowAll(plotshown);
          if not plotshown then
            Basic1Alert('No plots have been drawn.', 1);
        end;
      6 :
        MoveBack;
      7 :  { close front }
        CloseFront;
      otherwise
        ;
    end;{ case }
  end;

{----------------------SaveSimpBlock   procedure--------------------}
  procedure SaveSimpBlock (blockinH : bksHdl);
    var
      blockstosave, counter : integer;
  begin
    with filedata do
      begin
        numHandles := numHandles + 1;
        hItem[numHandles] := Handle(blockinH);
        numHandles := numHandles + 1;
        hItem[numHandles] := Handle(blockinH^^.subgrp);
        with blockinH^^.subgrp^^ do
          begin
```

```
        blockstosave := fwdbks + backbks;     { find no of blocks in group  }
        for counter := 1 to 5 do
          begin
            if bksused[counter]^^.used then      { is this block used }
              begin
                if bksused[counter]^^.simplified then { it is also simplified }
                  SaveSimpBlock(bksused[counter])
                else                    { it's not simplified so save it }
                  begin
                    numHandles := numHandles + 1;
                    hItem[numHandles] := Handle(bksused[counter]);
                  end;  { else block not simp so save it }
              end;  { if used }
          end;{ for }
      end;  { with blockinH.subgrp }
    end;{ with filedata  }
  end;

{------------------------SaveFile    procedure-----------------------}
  procedure SaveFile;
    var
      err : OSErr;
  begin
    InitDataList(filedata);
    SaveSimpBlock(sysblockH);
    err := WriteData(filedata, 'MCAD', 'CADD', savereply);
    if (err <> noErr) then
      Basic1Alert('Save unsuccessful. An error occurred while saving the file.', 2);
  end;

{------------------------GetBlockGroup    procedure-------------------}
  procedure GetBlockGroup (var blockinH : bksHdl);
    var
      newgroupH : grpHdl;
      newblockH : bksHdl;
      numbks, counter : integer;
  begin
    with newdata do
      begin
        numHandles := numHandles + 1;
        newgroupH := grpHdl(hItem[numHandles]);       { get the group }
        with newgroupH^^ do
          begin
            numbks := fwdbks + backbks;
            ownHdl := newgroupH;
            blockinH^^.subgrp := newgroupH;
            masterblock := blockinH;
            if numHandles = 2 then
              sysgroupH := newgroupH;
            for counter := 1 to numbks do         { get blocks in group }
              begin
                numHandles := numHandles + 1;
                bksused[counter] := bksHdl(hItem[numHandles]);
```

```
            bksused[counter]^^.fromgrpHdl := newgroupH;
            if bksused[counter]^^.simplified then   { this block is simplified }
              GetBlockGroup(bksused[counter]);
          end;{ for}
        if numbks < 5 then
          begin
            for counter := numbks + 1 to 5 do
              begin
                bksused[counter] := BksHdl(NewHandle(Sizeof(block)));
                bksused[counter]^^ := noblock;
                bksused[counter]^^.fromgrpHdl := newgroupH;
              end;  { for  }
          end;{ if bksused < 5  }
      end;  { with newgroupH  }
    end;  { with new data  }
  end;

{------------------------OpenFile   procedure-----------------------}
  procedure OpenFile;
    var
      err : OSErr;
      tempsysblock : bksHdl;
      temp1size, temp2size : size;
  begin
    temp1size := MaxMem(temp1size);
    InitDataList(newdata);
    savereply.good := anoldfile;
    err := ReadData(newdata, 'MCAD', 'CADD', savereply);
    if (err = noErr) then
      begin
        newdata.numHandles := 1;
        tempsysblock := bksHdl(newdata.hItem[1]);       { get sysblock  }
        sysblockH^^ := tempsysblock^^;       { get sysblock  }
        GetBlockGroup(sysblockH);
      end { open it}
    else
      Basic1Alert('An error occured while loading the selected file. No file has been opened.', 2);
  end;

end.  { unit  }
```

```
unit AllClose;
interface
  uses
    XTTypeDefs, CADGlobals, Extender1;

  procedure CloseAll;
implementation
  procedure CloseTheWindows;
  begin
    KillWindow(bodePtr);
    KillWindow(rootPtr);
    KillWindow(nyqPtr);
    KillWindow(timePtr);
  end;
  procedure CloseAll;
  begin
    CloseTheWindows;
    CloseResFile(fRefNum);
  end;

end.
```

LIST OF REFERENCES

1. Kuo, B. C. Automatic Control Systems, 4th ed. , p 563, Prentice-Hall, Inc, 1982.

2. Wood, R. L. Microcomputer Based Linear System Design Tool, M.S.E.E. thesis, Naval Postgraduate School, Department of Electrical and Computer Engineering, Monterey, Ca, Dec. 1986

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Commander Naval Weapons Canter China Lake, California 93555	1
3.	Chief of Naval Operations Attn: Code OP-03 Washington, D.C. 20350	1
4.	Director Naval Research Laboratory Washington, D.C. 20375	1
5.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
6.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
7.	Professor G. J. Thaler, Code 62Tr Department of Electrical Engineering Naval Postgraduate School Monterey, California 93943	15
8.	Assoc. Professor Daniel L. Davis, Code 52Dv Department of Electrical Engineering Naval Postgraduate School Monterey, California 93943	1

9. Professor H. A. Titus, Code 62Ti — 1
Department of Electrical Engineering
Naval Postgraduate School
Monterey, California 93943

10. Asst. Professor Jeffrey Burl, Code 62Bl — 1
Department of Electrical Engineering
Naval Postgraduate School
Monterey, California 93943

11. Lieutenant Kenneth MacDonald — 2
c/o. Cdr Donald A. MacDonald
420 Westcrest Drive
Kerrville, Texas 78028

12. Lieutenant Roy L. Wood — 1
Route 1, Driftwood Cove #20
Jefferson, Texas 75657

315 · 890

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

Thesis
M18327 MacDonald
c.1 MacCAD computer aided design tool for system analysis.

Thesis
M18327 MacDonald
c.1 MacCAD computer aided design tool for system analysis.

AMERICAN BINDERY
TOPEKA KANSAS

thesM18327
MacCAD computer aided design tool for sy
3 2768 000 77010 1
DUDLEY KNOX LIBRARY